# Knowledge Discovery e Data Mining: Concetti, Algoritmi e Sistemi

**Clara Pizzuti e Domenico Talia**
*ISI-CNR*
c/o DEIS
Univ. della Calabria
87036 Rende (CS), Italy

July 7, 1997

**Sommario**

*Le attività di ricerca e sviluppo nel settore del knowledge discovery e data mining riguardano lo studio e la definizione di tecniche, metodi e strumenti per l'estrazione di informazioni utili e non esplicitamente disponibili da grandi volumi di dati. Le tecniche di data mining hanno avuto origine dall'applicazione delle tecniche di analisi statistica e di machine learning per l'estrazione di informazioni da basi di dati. Negli ultimi anni, nuove tecniche e nuove algoritmi sono stati sviluppati. Questo rapporto presenta una introduzione agli algoritmi ed alle tecniche di data mining e knowledge discovery in basi di dati e descrive alcuni sistemi di data mining. Lo scopo è quello di fornire un'introduzione alle tecniche di data mining e di mostrare come le applicazioni di data mining e knowledge discovery nelle basi di dati possano portare significativi benefici per l'analisi intelligente di grandi moli di dati.*

**Abstract**

*Research and development work in the area of knowledge discovery and data mining concerns the study and definition of techniques, methods, and tools for the extraction of novel, useful and not explicitely available patterns from large volumes of data. Data mining techniques originated from the usage of statistical analysis and machine learning techniques for the mining of patterns from databases. In the last few years new techniques and algorithms have been designed. This report gives an introduction to algorithms and techniques for knowledge discovery and data mining in databases. Further, it describes few significant data mining systems. The goal is to give an introduction to data mining and show how the use of knowledge discovery and data mining techniques in databases might bring significant benefits for the intelligent analysis of large volumes of data.*

# 1 Introduzione

Negli ultimi dieci anni vi è stata una crescita esplosiva nelle nostre capacità sia di generare che di collezionare dati. I progressi nella tecnologia di memorizzazione dei dati basati su dispositivi di memoria sempre piu veloci e con maggiore capacità ad un costo inferiore, ci permettono di archiviare questo enorme flusso di dati. A titolo di esempio, in figura 1 è mostrata la crescita dell'ammontare di dati memorizzati in gigabytes in calcolatori di tipo mainframe con la previsione fino al 1998.

In questi ultimi anni sono state realizzate numerose applicazioni che gestiscono gigabytes di dati. Ad esempio, la più grande catena di negozi negli Stati Uniti, la Wal-Mart, gestisce 20 milioni di transazioni al giorno utilizzando una base di dati di oltre 3.5 terabytes di memoria [1]. Questo non è l'unico esempio. Vi sono database scientifici con una quantità enorme di dati. Un progetto sul genoma umano ha collezionato gigabytes di dati sul codice genetico umano [11], un database contenente un catalogo degli oggetti celesti provenienti da un osservatorio astronomico consiste di miliardi di immagini [12], il sistema di osservazione terrestre dai satelliti orbitanti costruito dalla NASA genera 50 gigabytes di immagini all'ora [45].

Questi volumi di dati chiaramente rendono poco utili i metodi tradizionali di analisi. Tali metodi, infatti, possono creare dei rapporti informativi su tali dati ma non sono in grado di analizzarne il contenuto e focalizzare l'attenzione sulle informazioni interessanti che essi contengono. E' stato stimato che trovare delle relazioni significative in un solo terabyte di dati, che è equivalente a due milioni di libri, richiederebbe la vita intera di una persona. Esiste pertanto una esigenza significativa per una nuova generazione di tecniche e strumenti capaci di assistere gli utenti in modo intelligente ed automatico nell'analizzare queste "montagne" di dati allo scopo di trovare conoscenza utile. Queste tecniche e strumenti sono argomento di studio nel campo *Scoperta di Conoscenza nelle Basi di Dati (Knowledge Discovery in Databases)*. Un *Sistema di Scoperta della Conoscenza* è un sistema che trova conoscenza che precedentemente non aveva, cioé che non era implicita nei suoi algoritmi o esplicita nella sua rappresentazione della conoscenza di dominio. Il termine *Knowledge Discovery in Databases*, in breve *KDD*, è stato coniato nel 1989 per riferirsi all'intero processo di scoperta della conoscenza nei dati e per enfatizzare l'applicazione ad alto livello di particolari metodi di estrazione dei dati, noti sotto il nome di *data mining*. Pertanto *KDD* è considerato l'intero processo, tipicamente interattivo ed iterativo, di trovare ed interpretare relazioni tra dati che coinvolge ripetute applicazioni di metodi o algoritmi specifici di data mining e l'interpretazione delle relazioni generate da questi algoritmi.

Le tecniche di data mining hanno un ruolo molto importante nel processo di KDD per scoprire ed estrarre in modo automatico infomazioni non immediatamente visibili da grandi moli di dati [15]. Come accennato in precedenza, i fattori principali che hanno contribuito allo sviluppo del data mining sono:

- le grandi accumulazioni di dati in formato elettronico,
- il data storage poco costoso, e
- i nuovi metodi e tecniche di analisi.

Naturalmente un aspetto fondamentale del processo di KDD ed in particolare del data mining è il fatto che per ottenere una migliore generalizzazione è necessario avere una migliore conoscenza. Miglioramenti nella conoscenza possono essere ottenuti: raccogliendo un maggior numero di informazioni, definendo modelli più complessi, oppure attraverso una analisi teorica più approfondita [5].

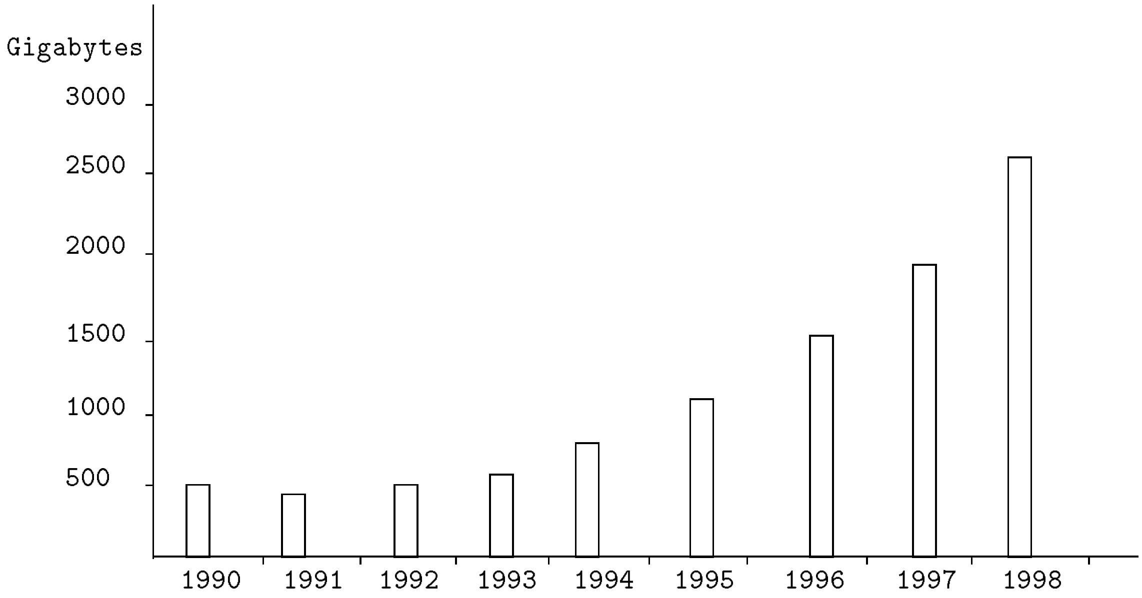



Figure 1: Crescita dell'ammontare di dati memorizzati in gigabytes in calcolatori di tipo mainframe con la previsione fino al 1998 (fonte International Data Corp.).

Come si è accennato, esistono oramai database composti da milioni di record la cui ricerca richiede algoritmi efficienti e con alte prestazioni. Per questo motivo l'utilizzo di tecniche avanzate assume una grande importanza per realizzare sistemi di knowledge discovery e data mining in database. Nel presente lavoro, dopo una introduzione ai concetti di base di $KDD$ e data mining, sono descritti i vari modelli teorici di rappresentazione usati nei sistemi data mining. Quindi sono presentati alcuni sistemi data mining rappresentativi dei diversi approcci finora usati. Infine, nell'ultima parte del lavoro sono elencati una serie di sistemi ed applicazioni di data mining paralleli. Infatti, siccome gli algoritmi di data mining richiedono l'analisi di grandi moli di dati, l'uso di algoritmi e architetture parallele possono portare notevoli vantaggi in termini di efficienza.

## 2 Il Sistema KDD

Un sistema $KDD$ è composto da un insieme di componenti che tutte insieme possono identificare ed estrarre relazioni dai dati memorizzati nella base di dati che siano nuove, utili ed interessanti. Un sistema $KDD$ ideale deve includere le seguenti componenti:

- **Controllore** controlla l'invocazione e la parametrizzazione delle altre componenti. Le decisioni del controllore sono basate sulla conoscenza del dominio e sulle informazioni fornite dall'utente. In alcuni sistemi dove il compito di scoperta è ben definito e statico [37], il controllore può semplicemente eseguire una sequenza predefinita di operazioni. In sistemi più versatili il controllore può assumere una maggiore responsabilità di decisione. Nella maggior parte dei sistemi, comunque, nel fare decisioni è richiesta la partecipazione dell'utente finale.

- **Interfaccia alla base di dati** genera e processa le richieste alla base di dati. I dati, infatti, sono estratti dalla base di dati utilizzando delle query. Una query tipica consiste di un certo numero di campi da ritrovare da un insieme di tabelle rispetto a determinati vincoli. La maggior parte dei sistemi relazionali di basi di dati supporta il linguaggio di interrogazione $SQL$.

- **Base di conoscenza** deposito di informazione specifica del dominio applicativo. Mentre un dizionario dati contiene informazioni sulla forma ed il contenuto della base di dati, specificando i nomi dei campi, i tipi e alcuni semplici vincoli sui valori che possono assumere, esistono ulteriori informazioni sulla struttura dei dati, vincoli tra diversi campi, conoscenza specifica sul dominio applicativo che può assumere forme diverse, ad esempio elenco di campi ritenuti importanti, gerarchie di generalizzazione, etc. Lo scopo principale della conoscenza di dominio è quello di guidare la ricerca di relazioni interessanti. Questo lo si può ottenere focalizzando l'attenzione su porzioni di dati, influenzando gli algoritmi di estrazione ed assistendo nella valutazione delle relazioni scoperte. È comunque da mettere in evidenza che, se da una parte la conoscenza sul dominio permette di fare scoperte rilevanti efficientemente, dall'altra preclude la possibilità di scoprire relazioni potenzialmente utili, in quanto alcune porzioni dello spazio di ricerca non sono considerate affatto. Esiste quindi uno scambio tra efficienza e completezza.

- **Focus** determina le porzioni dei dati da analizzare. Questa componente determina quali dati dovrebbero essere ritrovati dalla base di dati, quindi richiede la specifica di quali tabelle accedere, quali campi dovrebbero essere ritrovati e quali e quanti fatti dovrebbero essere restituiti. Per fare questa operazione la componente di focalizzazione ha bisogno di informazione dettagliata sulla struttura delle tabelle nella base di dati, deve sapere quali campi sono appropriati per il compito da svolgere, se fa campionamento sui dati deve avere una tecnica per selezionare casualmente i dati e deve, infine, conoscere l'input richiesto dagli algoritmi di estrazione per formattare i risultati opportunamente. La tecnica più comune di focalizzazione consiste nell'identificare gli attributi rilevanti. La limitazione nel numero di campi può comunque non essere sufficiente per restringere l'insieme dei dati su cui gli algoritmi di estrazione devono lavorare, in tal caso si deve selezionare un campione di fatti in maniera casuale.

- **Estrazione di pattern** una collezione di algoritmi di estrazione di pattern, dove il termine pattern si riferisce ad una qualsiasi relazione esistente tra elementi di una base di dati. Gli algoritmi usati per questo scopo costituiscono il *nucleo* di un sistema di scoperta. Diversi algoritmi basati su tecniche di *machine learning* e *analisi statistica* sono stati inclusi nei sistemi $KDD$. Una descrizione estesa di tali tecniche sarà data nel prossimo paragrafo.

- **Valutazione** valuta l'interessantezza e l'utilità dei pattern estratti. La componente di valutazione determina il relativo grado di interesse delle relazioni ritrovate e decide quali relazioni presentare ed in quale ordine.

## 2.1 Definizioni

Prima di iniziare ad approfondire la ricerca svolta nel campo del $KDD$ diamo alcune definizioni dei concetti [14] che verranno utilizzati nel seguito. Iniziamo proprio con la definizione di Knowledge Discovery nelle basi di dati.

*Knowledge Discovery nelle basi di dati* è il processo non triviale di identificare delle relazioni tra i dati che siano *valide, nuove, potenzialmente utili e comprensibili.*

Esaminiamo ognuno di questi temini in dettaglio.

- *Dati* i *dati* sono un insieme di fatti $F$, cioè dei casi in una base di dati.

- *Relazione* una *relazione* (o *pattern*) e' una espressione $E$ in un linguaggio $L$ che descrive fatti in un sottoinsieme $F_E$ di $F$. $E$ è chiamata relazione se è più *semplice*, rispetto ad un criterio di semplicità che verrà definito in seguito, della enumerazione di tutti i fatti di $F_E$.

- *Processo* un *processo* in $KDD$ è un insieme di attività che coinvolgono preparazione dei dati, ricerca di relazioni, valutazione della conoscenza e raffinamento della conoscenza scoperta. Il processo si assume che sia non banale, cioè che abbia un certo grado di autonomia di ricerca.

- *Validità* le relazioni scoperte dovrebbero essere valide su dati nuovi con qualche grado di certezza. Una misura della certezza viene definita come un funzione $C$ che associa espressioni di $L$ ad uno spazio metrico $M_C$ parzialmente o totalmente ordinato. Ad una espressione $E$ in $L$ su un sottoinsieme $F_E \subset F$ si può assegnare una misura di certezza $c = C(E, F)$.

- *Novità* le relazioni sono nuove (almeno per il sistema). La novità può essere misurata rispetto ai cambiamenti nei dati (confrontando valori correnti con quelli precedenti o quelli attesi) o nella conoscenza (cioè come una nuova scoperta è collegata a quella precedente). In generale si assume che la novità sia misurata da una funzione $N(E, F)$ che può essere sia una funzione booleana che un grado di novità.

- *Patenzialmente utile* le relazioni dovrebbero condurre a delle azioni utili, dove l'utilità è misurata da una funzione di utilità $U$ che associa espressioni di $L$ ad uno spazio metrico $M_U$, cioè $u = U(E, F)$.

- *Comprensibilità* uno degli obiettivi di $KDD$ è quello di rendere le relazioni scoperte comprensibili agli utenti per facilitare una migliore comprensione dei dati coinvolti. Pertanto viene definita una misura di semplicità della relazione attraverso una funzione $S$ che associa espressioni di $L$ ad uno spazio metrico $M_S$, cioè $s = S(E, F)$.

Un'altra nozione importante è il *l'interessantezza* della relazione che combina insieme tutti i parametri sopra definiti e viene misurata con una funzione da espressioni di $L$ in uno spazio metrico $M_I$, $i = I(E, F, C, N, U, S)$.
Si può a questo punto dire che cosa si intende per conoscenza in un processo di $KDD$.

*Conoscenza* una relazione $E \in L$ è chiamata conoscenza se, per qualche soglia specificata $i \in M_I$, $I(E, F, C, N, U, S) > i$.

Si noti che questa definizione di conoscenza non è assoluta ma dipende dalla scelta sia delle funzioni che delle soglie, a seconda del dominio applicativo, da parte di un utente.

*Data Mining* è un passo nel processo $KDD$ che consiste di particolari algoritmi di data mining che, sotto alcune limitazioni accettabili di efficienza computazionale, produce una particolare enumerazione dei pattern $E_j$ su $F$.

Lo spazio dei possibili pattern è spesso infinito e la loro enumerazione richiede una qualche forma di ricerca in questo spazio. I vincoli di efficienza computazionale pongono dei limiti piuttosto severi sul sottospazio che l'algoritmo può esplorare.

Il *Processo KDD* è il processo di usare metodi (algoritmi) di data mining per estrarre (identificare) quella che viene ritenuta conoscenza in accordo a delle misure e soglie specificate, usando la base di dati $F$ insieme ad attività di preprocessamento, sottocampionamento e trasformazioni di $F$.

La componente di data mining del processo $KDD$ riguarda principalmente il modo in cui i pattern sono estratti dai dati ed enumerati. La scoperta di conoscenza coinvolge la valutazione e la possibile interpretazione delle relazioni trovate per decidere cosa costituisce conoscenza e cosa non costituisce conoscenza.

Nel prossimo paragrafo affronteremo in dettaglio i metodi sviluppati per l'estrazione di informazione dai dati.

## 3 Data Mining

Data mining è una metodologia di risoluzione di problemi il cui scopo è di trovare una descrizione matematica o logica, eventualmente complessa, di relazioni e regolarità in un insieme di dati. Il processo di data mining, quindi, coinvolge la definizione di un modello dei dati a disposizione. Tale modello gioca il ruolo di conoscenza inferita. In letteratura sono stati definiti numerosi algoritmi di data mining da diversi campi di ricerca, tra cui statistica, machine learning, pattern recognition, basi di dati, ognuno dei quali utilizza tecniche specializzate per il rispettivo campo di applicazione.

Una base di dati può essere considerata come un deposito affibabile di informazione. Uno degli scopi principali di tale deposito consiste nel ritrovamento efficiente dell'informazione. Tale informazione non è semplicemente l'insieme dei dati memorizzati, ma più in generale è l'informazione che si può inferire dalla base di dati. Da un punto di vista logico esistono essenzialmente due principali tecniche di inferenza la *deduzione* e l'*induzione* [1] .

La *deduzione* permette di inferire informazione che sia conseguenza logica della informazione contenuta nella base di dati. Molti sistemi di gestione di basi di dati (DBMS), quali i sistemi relazionali, mettono a disposizione dei semplici operatori di deduzione dell'informazione, quale ad esempio l'operatore di join. In questo campo, comunque, vi è stato un notevole sviluppo per la realizzazione di basi di dati deduttive che arricchiscono un DBMS con capacità di deduzione logica (*KBMS, Knowledge Base Managment Systems*).

L'*induzione* permette di inferire informazione che è una generalizzazione dell'informazione contenuta nella base di dati. Questo tipo di informazione costituisce *conoscenza*, cioè delle affermazioni generali su proprietà degli oggetti. Quindi il processo induttivo cerca delle regolarità nella base di dati, cioè combinazioni di valori di certi attributi che siano condivisi da tutti i fatti della base di dati. Queste regolarità possono essere formulate in termini di regole in modo che il valore di un attributo possa essere previsto in termini di altri attributi.

La differenza sostanziale tra deduzione e induzione è che mentre la deduzione inferisce sempre dati che possono essere dimostrati validi all'interno della base di dati, purchè ovviamente la base di dati sia corretta, l'induzione inferisce asserzioni che sono solo supportate dalla base di dati ma che potrebbero non essere valide in generale. Pertanto uno dei compiti principali dell'induzione consiste nel selezionare le regole più plausibili che riducano al minimo gli errori nel caso siano sperimentate su fatti precedentemente sconosciuti.

Sebbene molti $DBMS$ supportino capacità di inferenza deduttiva ($KBMS$), ancora pochi supportano capacità di inferenza induttiva. Esiste quindi un crescente interesse nell'estendere i $DBMS$ con capacità di inferenza induttiva, ovvero con capacità di estrazione di conoscenza utile contenuta implicitamente nella base di dati che è appunto l'obiettivo principale del data mining [21].

L'apprendimento (o learning) induttivo, la cui automazione costituisce argomento di ricerca

[1] Esiste anche un altro tipo di inferenza, l'*abduzione* [4], che inferisce ipotesi che spiegano eventi osservati. Per una discussione sulla relazione tra induzione ed abduzione vedere [8].

nell'area del *machine learning*, può essere considerato come il tentativo, da parte di esseri intelligenti (spesso chiamati *sistema cognitivo*), di capire l'ambiente circostante attraverso un visione semplificata di tale ambiente consistente nella definizione di un *modello*. Durante la fase di apprendimento il sistema cognitivo osserva il suo ambiente e riconosce le analogie tra gli oggetti e gli eventi del suo ambiente. Quindi raggruppa oggetti simili in classi e costruisce regole per predire il comportamento dei componenti delle classi. Il processo di data mining può quindi essere visto come un processo di apprendimento induttivo quando l'ambiente circostante è una base di dati.

In generale, si può dire che gli algoritmi di data mining possono essere visti come la composizione di alcune tecniche e principi di base. In particolare, gli algoritmi di data mining sono costituiti dalla combinazione di tre componenti il *modello*, il *valutazione del modello* e il *metodo di ricerca*.

**Il modello** un modello, in generale, è un' astrazione della realtà di interesse. Per la definizione di un modello, nel data mining si devono tenere in considerazione due fattori la *funzione* del modello e la sua *rappresentazione*. La funzione, in generale, può essere di due tipi: *prescrizione* e *descrizione*. La *prescrizione* coinvolge l'utilizzo di alcune variabili o campi per predire valori sconosciuti o futuri di altre variabili di interesse ed è uno degli obiettivi principali delle applicazioni di machine learning e pattern recognition. La *descrizione* si focalizza nel trovare delle relazioni interpretabili dagli utenti che descrivano i dati. In ogni caso, i tipi di funzione più comuni realizzati negli algoritmi di data mining includono

- *Classificazione* classifica un insieme di dati in una o più classi predefinite. Quindi la classificazione è in grado di predire l'appartenenza di un dato ad una certa classe. Questo tipo di funzione verrà trattato più ampiamente nel prossimo paragrafo.

- *Regressione* associa un insieme di dati ad una variabile e predice il valore di tale variabile. Le applicazioni della regressione sono molte, ad esempio stimare la probabilità che un paziente possa ammalarsi in base ai risultati di esami diagnostici, predire il margine di vittoria di una squadra sportiva in base ai risultati ed ai dati tecnici degli incontri precedenti.

- *Clustering* cerca di identificare un insieme finito di categorie o raggruppamenti (cluster) per descrivere i dati. Le categorie possono essere mutuamente esclusive ed esaustive, oppure consistere di una rappresentazione più ricca quale categorie gerarchiche o che si sovrappongono. Esempi di applicazioni di clustering in *KDD* riguardano la scoperta di sottopopolazioni omogenee di consumatori in un database di vendite commerciali. Diversamente dalla classificazione in cui le classi sono predefinite, nel clustering le classi devono essere determinate dai dati cercando dei raggruppamenti basati su metriche di similarità tra i dati.

- *Summarization* fornisce una descrizione compatta di un sottoinsieme dei dati. Un esempio è la tabulazione della deviazione media e standard di ogni campo. Funzioni più complicate coinvolgono regole riassuntive, scoperta di relazioni funzionali tra variabili. Le tecniche riassuntive sono spesso usate nell'analisi interattiva di dati e nella generazione automatica di rapporti.

- *Modellazione delle dipendenze* consiste nel trovare un modello che descriva dipendenze significative tra variabili. I modelli di dipendenze sono a due livelli: il livello strutturale del modello specifica, spesso in forma grafica, quali variabili sono localmente dipendenti da ogni altra, il livello quantitativo specifica la potenza delle dipendenze usando qualche scala numerica.

- *Analisi dei legami* determina relazioni tra campi in una base di dati con lo scopo di derivare correlazioni multiple che soddisfino delle soglie specificate. Ad esempio, le regole associative descrivono quali prodotti sono venduti comunemente con altri prodotti in negozi alimentari.

- *Scoperta di episodi:* ricerca relazioni tra sequenze temporali di eventi. Ad esempio, se si verifica un dato evento $e_1$ allora entro un intervallo di tempo $t$ si verificherà un determinato evento $e_2$ con probabilità $p$.

La *rappresentazione* del modello è il linguaggio adottato per descrivere i pattern che si possono scoprire. La rappresentazione del modello determina sia la flessibilità del modello nel rappresentare i dati che la interpretabilità del modello in termini umani. Tipicamente i modelli più complessi possono rappresentare meglio i dati ma essere più difficili da capire. Tra i modelli di rappresentazione più noti ricordiamo alberi e regole di decisione, reti neurali, algoritmi genetici, metodi basati su esempi, modelli di dipendenza probabilistica e modelli relazionali.

**Valutazione del Modello** determina quanto un particolare modello e i suoi parametri soddisfano i criteri del processo $KDD$. La valutazione del modello quindi stabilisce il criterio di preferenza di un modello rispetto ad un altro in base a criteri statistici o logici.

**Metodo di Ricerca** un metodo di ricerca consiste di due componenti, *ricerca dei parametri* e *ricerca del modello.* Nel primo caso l'algoritmo deve cercare i parametri che ottimizzano i criteri di valutazione del modello, in base ai dati osservati ed una rappresentazione fissata del modello. La ricerca del modello la si può quindi considerare come un processo iterativo sul metodo di ricerca dei parametri. Si considera una famiglia di modelli e ognuno di essi viene valutato attraverso il metodo di ricerca dei parametri. Poichè lo spazio dei possibili modelli è in generale molto grande, le implementazioni dei metodi di ricerca utilizzano delle tecniche di ricerca euristica.

I metodi di data mining sviluppati per la definizione e la scelta di un modello seguono essenzialmente due approcci l'approccio basato su metodi statistici [9, 22], che storicamente sono stati usati per analizzare i dati con l'obiettivo di trovare correlazioni e dipendenze tra insiemi di dati, e l'approccio basato su tecniche di machine learning. In questo rapporto focalizzeremo la nostra attenzione essenzialmente su questo secondo tipo di approccio, sviluppando, inoltre, le componenti descritte che costituiscono un metodo di data mining.

# 4 Data Mining come Apprendimento Induttivo

Negli ultimi decenni vi è stato un notevole lavoro di ricerca in diverse discipline correlate, che vanno dalla psicologia all'intelligenza artificiale, con lo scopo di sviluppare modelli generali della conoscenza ed in particolare dell'apprendimento. Questo sforzo ha prodotto un certo numero di tecnologie interessanti, in particolare le tecniche per la rappresentazione della conoscenza e di apprendimento hanno permesso la realizzazione di importanti applicazioni per la rappresentazione e l'analisi dei dati. In questo paragrafo è descritta più formalmente la metodologia del data mining dal punto di vista del machine learning.
Esistono due tecniche principali di learning il *supervised* e l' *unsupervised learning.*

Nel *supervised learning* esiste un insegnante esterno che definisce le classi e fornisce gli esempi di ogni classe al sistema cognitivo. Il compito del sistema quindi consiste nello scoprire proprietà comuni negli esempi di ogni classe, cioè deve fornire una *descrizione* della classe. Questa tecnica è anche nota come *apprendimento da esempi.* Una classe, insieme alla sua descrizione, forma una *regola di classificazione* del tipo **if** *description* **then** *class* che può essere usata per predire la classe di un nuovo oggetto.

Nell' *unsupervised learning* è il sistema che deve scoprire le classi stesse, in base a proprietà comuni degli oggetti, senza l'aiuto di un insegnante. Il sistema deve osservare gli esempi e riconoscere le relazioni tra di essi. Questa tecnica è pertanto anche nota come *apprendimento da osservazione e scoperta.* Il risultato di questo processo è un insieme di descrizioni di classi, una per ogni classe scoperta, che insieme coprono tutti gli oggetti dell'ambiente. Queste descrizioni formano un sommario ad alto livello degli oggetti nell'ambiente.

Il processo di costruzione di una descrizione di una classe di ogetti è un processo iterativo in cui tra tutte le possibili descrizioni si cerca la migliore. Una ipotesi, cioè una descrizione iniziale, viene formulata e valutata in base a qualche funzione di qualità. Quindi questa ipotesi è accettata, rifiutata o migliorata finchè non si trova una ipotesi corretta. Questo processo lavora su un insieme di esempi detto *insieme di addestramento (training set)* e non su tutto l'ambiente. Nel caso del data mining l'insieme di addestramento è un sottoinsieme dei fatti contenuti nella base di dati. Poichè la maggior parte dei DBMS sono dei sistemi relazionali, gli oggetti dell'ambiente sono rappresentati dalle tuple della base di dati. Un'assunzione che si fa comunemente nelle basi di dati è l' *Assunzione di Relazione Universale* [43] che consiste nel considerare la base di dati come un'unica tabella. Questa assunzione risulta molto utile perchè permette di avere una visione semplificata della base di dati. Essa ha come inconveniente, però, il fatto che molti dei valori nella tabella possono essere nulli o sconosciuti.
Definiamo ora formalmente cosa si intende per *training set* e *classe.*

**Training set.** Sia $A = \{A_1, \ldots, A_n\}$ un insieme di attributi con domini $Dom_1, \ldots, Dom_n$. Un *training set* è una tabella su $A$. Un esempio, o fatto, è una tupla del training set. L' *Universo* $U$ è tutta la relazione su $A$ cioè $U = Dom_1 \times \ldots \times Dom_n$. Quindi ogni training set è un sottoinsieme finito di $U$. Si assume che $U$ sia finito.

**Classe.** Una classe $C_i$ è un sottoinsieme del training set $S$, consistente di tutti gli oggetti che soddisfano la condizione di classe $cond_i$

$$C_i = \{o \in S \mid cond_i(o)\}$$

Gli oggetti che soddisfano la condizione $cond_i$ sono detti esempi *positivi* o *istanze* della classe $C_i$, quelli che non la soddisfano sono detti esempi *negativi.*

Apprendere regole di classificazione significa che il sistema deve trovare le regole che predicono la classe. Innanzitutto l'utente deve definire le condizioni per ogni classe e quindi partizionare $S$ in sottoinsiemi $C_1, \ldots, C_n$. Quindi il sistema di estrazione dei dati deve costruire le descrizioni $D_1, \ldots, D_n$ per queste classi. Le descrizioni più usate nei sistemi di data mining sono un sottoinsieme delle condizioni di selezione dell'algebra relazionale e vengono chiamate *set description.*

**Descrizione.** Sia $A$ un insieme di attributi. Una *descrizione elementare* è una formula $A_1 = c_1 \wedge \ldots \wedge A_n = c_n$ tale che

1. $A_i \in A$ e $i \neq j \rightarrow A_i \neq A_j$;

2. $c_i \in Dom(A_i)$

Un *pattern* o *descrizione* è una disgiunzione (non vuota) di descrizioni elementari. Una condizione $A_i = c_i$ è chiamata *condizione attributo-valore.* L'insieme di tutte le possibili descrizioni è chiamato *spazio delle descrizioni* ed è denotato da $D$.

Per convenienza notazionale, si permette che il valore $c_i$ di un attributo sia un insieme di valori $S_i \subseteq Dom(A_i)$, cioè si può scrivere

$$A_1 \in \{c_{1,1}, \ldots, c_{1,m_1}\} \wedge \ldots \wedge A_n \in \{c_{n,1}, \ldots, c_{n,m_n}\}$$

che è equivalente a

$$(A_1 = c_{1,1} \wedge \ldots \wedge A_n = c_{n,1}) \vee \ldots \vee (A_1 = c_{1,m_1} \wedge \ldots \wedge A_n = c_{n,m_n})$$

Tutti gli esempi che soddisfano una descrizione $D$ sono detti essere *coperti* da $D$. In altre parole, gli esempi contenuti in $\sigma_D(S)$, cioè la selezione relazionale degli elementi di $S$ che soddisfano $D$, sono gli esempi coperti da $D$.

**Regola.** Una *regola di classificazione* consiste di una *descrizione* $D$ e di un *simbolo di classe* $C$

$$\forall o \in U o \in \sigma_D(S) \rightarrow o \in C$$

o semplicemente *if D then C*, che dice che ogni oggetto che soddisfa $D$ appartiene alla classe $C$.

Il linguaggio set-description è un linguaggio molto semplice. In letteratura ne sono stati sviluppati altri basati sulla logica proposizionale, sulla logica del primo ordine, sulle reti semantiche e sulle reti neurali. Ognuno di questi formalismi verrà approfondito nei prossimi paragrafi. Prima, però, vogliamo mostrare come affrontare il problema di cercare la migliore descrizione possibile nello spazio di tutte le descrizioni, cioè il problema della valutazione del modello scelto. Poichè lo spazio delle possibili descrizioni è troppo vasto, molti sistemi di data mining scelgono una descrizione iniziale e iterativamente la modificano cercando di migliorarne la qualità. La scelta della descrizione iniziale e l'ordine con cui differenti alternative sono considerate costituisce la strategia con cui lo spazio di ricerca viene esplorato. La strategia di ricerca è un altro fattore importante nel processo di data mining.

# 5 Valutazione del Modello e Metodo di Ricerca

Dato un training set e delle classi definite dall'utente, un sistema di data mining può costruire molte descrizioni. Alcune di queste sono più corrette di altre, cioè alcune di esse classificano meglio gli esempi ancora sconosciuti. Una volta definita una misura di qualità di una descrizione, la costruzione di una descrizione può essere espressa come un problema di ricerca trovare la migliore descrizione nell'insieme delle descrizioni costruibili $D$. Si deve dare quindi una definizione di spazio di ricerca [29] e si devono specificare quali sono le strategie di esplorazione di questo spazio al fine di ottenere la descrizione migliore nel minor tempo possibile.

## 5.1 Spazio di ricerca.

Lo spazio di ricerca $< D, O, f >$ consiste di un insieme di descrizioni $D$, un insieme di operazioni $O$ su queste descrizioni ed una funzione di qualità $f$.

L'insieme di operazioni $O$ possono essere suddivise in *generalizzazione*, che indebolisce la descrizione, cioè la descrizione copre più oggetti, e la *specializzazione*, che rafforza la descrizione, cioè riduce la sua copertura.

L'applicazione di una operazione di generalizzazione ad una descrizione $D$ risulta in una nuova descrizione $D'$ che copre più oggetti, cioè $\sigma_D(S) \subseteq \sigma_{D'}(S)$. Più formalmente, una set-description $D = (A_1 \in S_1 \wedge \ldots \wedge A_i \in S_i \wedge \ldots \wedge A_n \in S_n)$ è *generalizzata* estendendo l'insieme dei valori di un particolare attributo $A_i$ a $S_i'$ dove $S_i \subset S_i' \subseteq Dom_i$. Un caso particolare di generalizzazione è l' operazione di eliminazione di una condizione in cui l'insieme $S_i$ è esteso all'intero dominio $Dom_i$.

L'operazione di specializzazione è l'inverso della generalizzazione, quindi un insieme $S_i$ è sostituito con $S_i'$ e $S_i \supset S_i'$.

La *funzione di qualità* assegna un valore ad ogni descrizione, indicandone la sua qualità. Una descrizione dovrebbe essere *valida* in generale, cioè dovrebbe classificare correttamente gli oggetti non ancora visti, e dovrebbe essere corretta rispetto alle classi definite dall'utente, cioè deve coprire tutti gli esempi positivi e nessuno dei negativi. La validità di una regola non può essere dimostrata in quanto non è possibile verificare la sua correttezza per tutte le possibili situazioni. Pertanto la possibilità che una descrizione sia più valida di un'altra viene misurata in base alla sua semplicità. La validità $f_v$ è più alta per descrizioni semplici. Per quanto riguarda la correttezza, invece, si definiscono i seguenti concetti.

**Accuratezza di Classificazione.** Si definisce *accuratezza della classificazione* la probabilità che la regola classifichi correttamente, cioè la probabilità che un oggetto coperto dalla descrizione appartenga effettivamente alla classe. L'accuratezza della classificazione è la porzione relativa di $\sigma_D(S)$ che è anche coperta da $C$

$$accuratezza\ di\ classificazione = \frac{\mid \sigma_D(S) \cap C \mid}{\mid \sigma_D(S) \mid}$$

**Copertura.** Si definisce copertura di una descrizione la probabilità che un oggetto arbitrario, appartenente alla classe $C$ sia coperto dalla descrizione $D$

$$copertura = \frac{\mid \sigma_D(S) \cap C \mid}{\mid C \mid}$$

**Regole Complete.** Se la copertura vale 1, si dice che la regola è *completa*, cioè qualsiasi oggetto appartenente alla classe è coperto dalla descrizione, $C \subseteq \sigma_D(S)$.

**Regole Deterministiche**. Se l'accuratezza della classificazione vale 1, la regola si dice *deterministica*, cioè la regola classifica sempre in maniera corretta. Qualsiasi oggetto coperto dalla descrizione appartiene alla classe, $C \supseteq \sigma_D(S)$.

**Regole Corrette.** Se sia l'accuratezza di classificazione che la copertura valgono 1, la regola si dice *corretta*.

Il criterio di correttezza $f_c$ assume il valore 1 se la descrizione è corretta, ovviamente il valore delle regole scorrette è inferiore ad uno.

## 5.2 Strategie di ricerca

Data una descrizione iniziale $D_1$, si possono applicare alcune delle operazioni di $O$, $o_1, \ldots, o_n$. L'applicazione di ognuna di queste operazioni $o_i$ genera $n$ nuove descrizioni $D_{2i}$. Reiterando questo procedimento si costruisce un grafo di ricerca in cui le descrizioni formano i nodi e le operazioni gli archi. Si deve quindi trovare una descrizione di qualità sufficiente, cioè un nodo foglia nel grafo nel minor tempo possibile. La costruzione di una regola si può caratterizzare come un processo di ricerca in cui si provano le diverse operazioni finchè una qualche sequenza di operazioni non produce una regola sufficientemente buona. Il modo in cui si cerca di costruire queste sequenze costituisce la strategia di ricerca. Esistono sostanzialmente due strategie di ricerca. La *ricerca irrevocabile* in cui ad ogni passo si seleziona una sola operazione senza possibilità di essere riconsiderata successivamente. In pratica si valuta un singolo cammino nel grafo. La *ricerca a tentativi* seleziona una operazione da applicare ma tale operazione può essere annullata se non porta a dei risultati significativi. Vi sono due tipi di strategie a tentativi. Nel *backtraking* ogni volta che si seleziona un'operazione questa può essere punto di backtracking per cui si può fare una scelta alternativa a partire da tale punto. Nella strategia *ricerca del grafo* il sistema memorizza l'intera porzione del grafo visitato fino a quel momento. La visita del grafo nella ricerca a tentativi può inoltre essere fatta in profondità oppure in ampiezza. Infine queste strategie di ricerca possono scegliere il successivo nodo da visitare in maniera casuale, senza alcun tipo di informazione che possa guidare la scelta, oppure in base ad una euristica.

# 6 Rappresentazione del modello

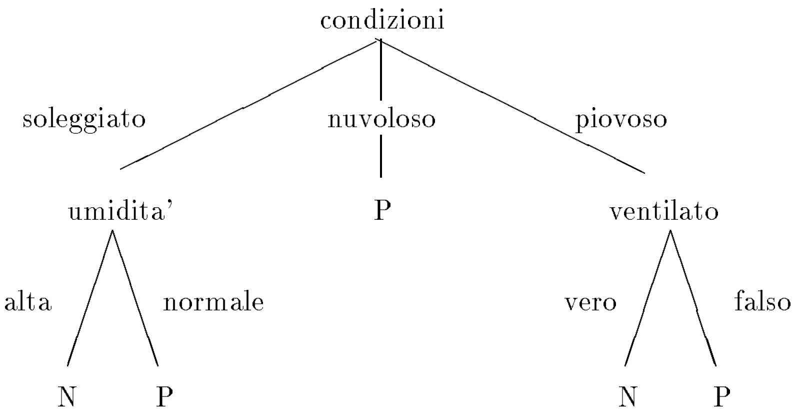
La rappresentazione del modello è il linguaggio con cui si descrivono i pattern che possono essere scoperti. Se il linguaggio di rappresentazione è troppo limitato l'accuratezza del modello sarà pure bassa. Nel seguito sono descritti i principali formalismi adottati.

## 6.1 Calcolo Proposizionale

Le rappresentazioni basate sul calcolo proposizionale usano formule logiche consistenti di condizioni attributo-valore. Ad esempio $(colore = rosso \vee colore = grigio) \wedge (forma = cerchio)$ è una formula in forma normale congiuntiva ($CNF$), cioè una congiunzione di clausole, dove ogni clausola è una disgiunzione di condizioni attributo-valore. Le set-description che abbiamo introdotto nel precedente paragrafo, in realtà sono delle $CNF$. La formula di prima può essere riscritta come $colore \in \{rosso, grigio\} \wedge forma \in \{cerchio\}$. Una rappresentazione alternativa è rappresentata dalle forme normali disgiuntive ($DNF$), una disgiunzione di termini, dove i termini sono congiunzioni di condizioni attributo-valore. È stato comunque dimostrato che le $CNF$ sono migliori delle $DNF$ in quanto le descrizioni generate sono più piccole delle rappresentazioni $DNF$ generate per lo stesso compito di apprendimento. C'è da notare che le rappresentazioni non sono realmente proposizionali in quanto coinvolgono una variabile che è l'oggetto stesso. Ad esempio, quando la formula di prima è usata in una regola di produzione, essa è una condizione su un oggetto $X$ del tipo $P(X) = (X.colore = rosso \vee X.colore = grigio) \wedge (X.forma = cerchio)$. Tra le rappresentazioni basate sul calcolo proposizionale si possono enumerare gli *alberi di decisione*, le *regole di produzione*, le *liste di decisione* e gli *insiemi di regole ripple-down*.

**Alberi di Decisione**. Gli alberi di decisione sono una delle tecniche più sviluppate per partizionare insiemi di oggetti in classi. Furono inizialmente introdotti nel sistema di apprendimento $ID3$ di Quinlan [31], uno dei primi sistemi di apprendimento realizzati. Un albero di decisione classifica esempi in un numero finito di classi. I nodi dell'albero sono etichettati con i nomi degli attributi, gli archi sono etichettati con i possibili valori dell'attributo e le foglie sono etichettate con le differenti classi. Un oggetto è classificato seguendo un cammino lungo l'albero costituito dagli archi corrispondenti ai valori degli attributi dell'oggetto.

Supponiamo di avere un training set consistente di tuple che descrivono le condizioni meteorologiche in un particolare momento. Le tuple contengono informazioni sulle condizioni del tempo, che possono essere *soleggiato*, *nuvoloso*, *piovoso*, sull'umidità, che può essere *alta* o *normale*, e sul vento, che può essere *presente* o *assente*. Alcuni oggetti sono esempi positivi di una classe $N$, altri sono esempi negativi. Il compito della classificazione consiste nel costruire un semplice albero che classifichi tutti gli esempi nel training set in modo corretto. Si potrebbe ad esempio costruire il seguente albero




**Regole di Decisione**. Uno dei principali svantaggi degli alberi di decisione è che essi tendono a crescere eccessivamente su applicazioni reali e quindi sono difficili da interpretare. È stato fatto un notevole sforzo di ricerca per trasformare gli alberi di decisione in altri tipi di rappresentazioni più semplici. Quinlan stesso [32] ha descritto una tecnica per generare regole di produzione da alberi di decisione. Questa trasformazione è utile per tre motivi diversi. Innanzitutto le regole di decisione sono un formalismo ampiamente utilizzato per rappresentare conoscenza, in secondo luogo esse sono facilmente interpretabili dagli esperti a causa della loro estrema modularità, ogni regola può essere capita senza alcun riferimento ad altre regole. Infine, l'accuratezza di classificazione di un albero di decisione può essere migliorata trasformando l'albero in un insieme di regole di produzione. L'albero di decisione precedente può, ad esempio essere trasformato nel seguente insieme di regole

**if** *condizioni* = *soleggiato* **and** *umidità* = *alta* **then** *class* = *N*
**if** *condizioni* = *piovoso* **and** *vento* = *vero* **then** *classe* = *N*
*classe di default* = *P*

**Liste di Decisione**. Le liste di decisione, proposte in [36], sono una generalizzazione degli alberi di decisione. Una lista di decisione è una lista di coppie

$$(\phi_1, C_1), \ldots, (\phi_r, C_r)$$

dove ogni $\phi_i$ è una descrizione elementare, ogni $C_i$ è una classe e l'ultima descrizione $\phi_r$ è la costante *vero*. La classe di un oggetto $o$ è $C_j$ se $j$ è l'ultimo indice di una descrizione $\phi_j$ che copre l'oggetto $o$. Tale indice esiste sempre in quanto l'ultimo termine è sempre vero, cioè la classe di default. Si possono pensare le liste di decisione come un formalismo per definire classi dando dei pattern generali con eccezioni. Le eccezioni corrispondono ai primi termini della lista, mentre gli ultimi termini sono le descrizioni più generali.

**Insiemi di Regole Ripple-down**. Una delle caratteristiche principale delle regole è che esse hanno tendenza ad avere eccezioni. Regole più realistiche hanno la forma *if* $\phi_i$ *then* $C_i$ *unless* $\phi_j$. Questa regola la si può rappresentare in una lista di decisione ponendo una regola di eccezione *if* $\phi_j$ *then* $C_j$ nella lista prima della regola *if* $\phi_i$ *then* $C_i$. Il principale svantaggio di usare le liste di decisione è che l'eccezione è globale, cioè qualsiasi oggetto per cui $\phi_j$ è vera gli verrà assegnata la classe $C_j$, mentre noi avremmo voluto che l'eccezione fosse locale alla regola *if* $\phi_i$ *then* $C_i$. Questa mancanza di località rende le liste difficili da capire, in quanto le regole nella lista sono significative solo nel contesto dato da tutte le regole precedenti. La rappresentazione di eccezioni in un modo più localizzato è realizzato degli insiemi di regole ripple-down [6].Queste regole consistono di condizioni ed eccezioni a queste condizioni che sono locali alle regole. Queste regole sono infatti degli if-then innestati del tipo

if $\phi_i$ then
  if $\phi_j$ then $C_j$
    else $C_i$
  else $C_j$

Poichè le eccezioni sono locali, non c'è necessità di ordinare le regole come nelle liste di decisione.

## 6.2 Calcolo dei predicati del primo ordine

I linguaggi di descrizione basati sul calcolo proposizionale sono stati applicati con successo in molti sistemi di apprendimento. Tali linguaggi però, poichè esprimono le relazioni scoperte in un formato del tipo attributo-valore, hanno il potere espressivo del calcolo proposizionale e pertanto presentano una serie di inconvenienti dovuti principalmente alla limitatezza del linguaggio. Questo tipo di rappresentazione infatti limita fortemente il formato delle relazioni che possono essere rappresentate. Non è possibile ad esempio rappresentare relazioni tra oggetti o attributi. Consideriamo la classe delle persone aventi stesso nome e cognome. In un linguaggio proposizionale l'unico modo di rappresentare questa classe consiste nell'enumerare tutte le persone con questa caratteristica. Con un linguaggio più potente sarebbe sufficiente dire che qualsiasi persona per cui $nome = cognome$ appartiene alla classe. È inoltre difficile incorporare la conoscenza sul dominio durante il processo di apprendimento. Una tecnica comunemente usata consiste nel considerare la conoscenza sul dominio come dei vincoli, generati dal sistema, sulle descrizioni. Poichè la conoscenza sul dominio è raramente completa e consistente, questi vincoli sono considerati troppo restrittivi. Inoltre i sistemi induttivi attuali costruiscono descrizioni nei limiti di un vocabolario di attributi fissato. Per molte applicazioni sarebbe utile aumentare l'insieme delle descrizioni che possono essere trovate e migliorare la comprensibilità della rappresentazione inventando predicati ausiliari.

Tutti questi inconvenienti possono essere superati addottando dei linguaggi di rappresentazione più potenti. Negli ultimi anni un numero crescente di sistemi di apprendimento ha iniziato ad utilizzare la *Logica del Primo Ordine* ($FOL$) per rappresentare la conoscenza da apprendere con lo scopo di costruire una definizione logica (intensionale) di una relazione a partire da tuple note appartenere o non appartenere alla relazione e con la possibilità di utilizzare altre relazioni facenti parte della conoscenza di dominio. Questa nuova area di ricerca è stata recentemente chiamata *Programmazione Logica Induttiva (ILP)* [28, 24, **?**, 7].

La *programmazione logica induttiva* tratta essenzialmente il problema di imparare *programmi logici* da esempi. Intuitivamente un esempio dovrebbe descrivere il comportamento ingresso/uscita di un programma in un caso specifico. Ad esempio, se consideriamo l'intersezione di due liste si avrebbe:

ingresso : [4,2,6],[6,5,2,8]
uscita : [2,6]

In programmazione logica questo è facilmente formalizzato richiedendo che gli esempi siano ground, cioè che non contengano variabili. L'intersezione delle due liste si può riscrivere

int([4,2,6],[6,5,2,8],[2,6]).

Gli esempi non solo descrivono il comportamento del programma, ma possono anche fornire informazione negativa, cioè quello che il programma non dovrebbe calcolare. In generale i sistemi di learning induttivo usano *esempi positivi* ed *esempi negativi*. Gli esempi positivi sono letterali ground considerati veri dall'utente, gli esempi negativi sono letterali ground che l'utente etichetta come falsi. Ad esempio il letterale int([4,2,7],[6,5,2,8],[2,6]) potrebbe essere un esempio negativo di intersezione. I programmi appresi non dovrebbero, ovviamente, eseguire computazioni che contraddicono gli esempi. Se consideriamo un insieme $E^+$ di esempi positivi ed un insieme $E^-$ di esempi negativi, questo concetto è reso preciso dalle seguenti definizioni:

Un programma logico $P$ è *completo* (rispetto ad $E^+$) se e solo se per tutti gli esempi $e \in E^+$, $P \vdash e$, dove $\vdash$ è la derivazione logica.

Un programma logico $P$ è *consistente* (rispetto ad $E^-$) se e solo se per nessun esempio $e \in E^-$, $P \vdash e$.

Ad esempio se:

$E^+ = \{int([4,2,7],[4],[4]), int([4,2,7],[],[]),$

$E^- = \{int([4,2,7],[6],[6])$ e

$P = int(X,Y,Y)$

allora $P$ è completo perchè deriva entrambi gli esempi positivi ma non è consistente perchè deriva anche l'esempio negativo. La formulazione base di un problema $ILP$ è la seguente:

**Dati:**

un insieme $\mathcal{P}$ di possibili programmi

un insieme $E^+$ di esempi positivi

un insieme $E^-$ di esempi negativi

un programma logico consistente $B$ tale che $B \not\vdash e^+$ per almeno un $e^+ \in E^+$

**Trovare:**

un programma logico $P \in \mathcal{P}$ tale che il programma $B, P$ sia completo e consistente.

È da notare che:

- Il programma logico $B$, che è dato come input al sistema di apprendimento, viene considerata la conoscenza di dominio; esso rappresenta una parte del programma desiderato che l'utente ha già a disposizione.

- Il programma $B, P$ è ottenuto aggiungendo le clausole di $P$ dopo quelle di $B$. Notazioni analoghe sono $B \cup P$ e $B \wedge P$.

- L'insieme $\mathcal{P}$ dei programmi logici che si possono imparare è chiamato *spazio delle ipotesi*, quindi $P$ viene anche chiamato *ipotesi*. Usualmente gli esempi, $B$ e $P$ devono soddisfare delle restrizioni sintattiche. Tali restrizioni sono chiamate *bias* del linguaggio. Un *bias*, quindi, non è altro che una descrizione dello spazio delle ipotesi contenente tutti i programmi logici che il sistema può imparare.

I sistemi di programmazione logica induttiva si possono suddividere in diverse categorie, a seconda dei diversi parametri considerati. Innanzitutto si deve fare una distinzione tra sistemi che apprendono un singolo concetto (predicato) e quelli che ne apprendono più di uno. Nel primo caso il programma logico indotto contiene clausole tutte con lo stesso simbolo di predicato (*single predicate learning*), nel secondo caso il simbolo di predicato nella testa non è sempre lo stesso (*multiple predicate learning*). Secondo, tali sistemi possono richiedere tutti gli esempi del training set all'inizio, in tal caso si parla di sistemi *empirici*, oppure possono accettare gli esempi uno alla volta, sistemi *incrementali*. Inoltre i sistemi possono essere *interattivi*, cioè si può supporre che abbiano a disposizione un oracolo con cui verificare la validità delle classificazioni fatte, o *non interattivi*. Un sistema $ILP$ può anche *inventare* nuovi predicati, cioè il programma generato contiene la definizione di almeno un predicato che non compare nè negli esempi nè nella conoscenza di dominio. Infine, un sistema $ILP$ può accettare una teoria iniziale, da considerare abbastanza vicina al risultato finale, e quindi, in base alle informazioni contenute negli esempi, la teoria viene modificata e riveduta durante il processo di apprendimento. Sebbene le caratteristiche elencate siano, in linea di principio, indipendenti, i sistemi $ILP$ sviluppati si pongono in due categorie: da una parte vi sono i sistemi empirici che apprendono un singolo predicato, dall'altra vi sono i sistemi interattivi, incrementali, revisori di teorie e che apprendono più predicati.

L' induzione, sotto un certo punto di vista, può essere considerata come l'opposta della deduzione. La distinzione tra regole di inferenza deduttive e induttive è anche legata alla nozione di *specializzazione* e *generalizzazione* nella inferenza induttiva: la deduzione corrisponde alla specializzazione mentre l'induzione corrisponde alla generalizzazione. In termini logici una ipotesi $\psi$ è più generale di una ipotesi $\phi$, rispetto ad una teoria $B$, se e solo se $B \wedge \psi \models \phi$, dove $\models$ è l'implicazione logica. Questa viene chiamata nozione *semantica* della generalità. Derivare $\phi$ da $B$ e $\psi$ è deduzione o specializzazione. Derivare $\psi$ da $\phi$ e $B$ è induzione o generalizzazione. In $ILP$ $\psi$ è il programma (l'ipotesi) $P$ che deve essere indotto dagli esempi e $\phi$ sono gli esempi positivi che devono essere derivati da $P$. Generalizzazione e specializzazione permettono di eliminare molte porzioni dello spazio di ricerca perchè:

- se $B \wedge P \not\models e^+$ allora nessuna specializzazione $P'$ di $P$ potrà derivare tutti gli esempi positivi; pertanto tutte le specializzazioni $P'$ possono essere tagliate dallo spazio di ricerca.

- se $B \wedge P \wedge e^- \models \Box$ dove $\Box$ denota la clausola vuota, allora tutte le generalizzazioni $P'$ di $P$ saranno inconsistenti con $B$; pertanto possono essere tagliate.

Queste due proprietà spiegano perchè molti approcci in $ILP$ sono o *dal generale verso lo specifico* oppure *dallo specifico verso il generale*. A causa della nozione data di generalizzazione, le regole di inferenza induttiva si possono ottenere invertendo le regole di inferenza deduttive. Due tecniche base sono state sviluppate, la tecnica di *generalizzazione* esamina lo spazio di ricerca in una maniera *bottom-up*: a partire dagli esempi di addestramento, visita lo spazio delle ipotesi usando operatori di generalizzazione. Questa tecnica è adatta per il learning interattivo e incrementale quando si hanno pochi esempi a disposizione. La tecnica di *specializzazione* visita lo spazio delle ipotesi *top-down* da descrizioni di concetti il più generale possibile verso lo specifico utilizzando operatori di specializzazione. Questa tecnica è più adatta per learning empirico in quanto la ricerca può essere guidata da euristiche. Una tecnica $ILP$ può quindi essere vista come una ricerca, spesso guidata da euristiche, che procede attraverso lo spazio delle ipotesi ben formate determinate dal bias del linguaggio, usando regole di generalizzazione e specializzazione. Gli stati nello spazio di ricerca sono descrizioni di concetti e l'obiettivo è di trovare uno o più stati che soddisfano qualche criterio di qualità (il criterio di qualità standard è completezza e consistenza). Considerando l' $ILP$ come ricerca, un learner $ILP$ pò essere ora descritto in termini della struttura del suo spazio di ricerca, della sua strategia di ricerca e delle euristiche di ricerca. Sono state definite due tecniche fondamentali di visita dello spazio di ricerca: metodi bottom-up e metodi top-down.

### 6.2.1 Metodi Bottom-up

Le tecniche di generalizzazione visitano lo spazio delle ipotesi in modo bottom-up. Si parte dalla clausola più specifica permessa dal bias del linguaggio che copre un dato esempio positivo e quindi si generalizza tale clausole finchè una sua ulteriore generalizzazione ricoprirebbe degli esempi negativi.

Per poter visitare lo spazio di ricerca dei programmi, è utile strutturare tale spazio introducendo un ordine parziale sull'insieme delle clausole basato sul concetto di $\theta$-sussunzione, che introduce la nozione *sintattica* di generalità, che è più facilmente trattabile, da un punto di vista computazionale, della nozione *semantica* corrispondente.

**$\theta$-sussunzione.** Siano $C$ e $D$ due clausole. La clausola $C$ *$\theta$-sussume* (o è una *generalizzazione* di) $D$ se esiste una sostituzione $\theta$, tale che $C\theta \subseteq D$, usualmente scritto come $C \leq D$. $C$ è la *generalizzazione meno generale (least general generalization, lgg)* di $D$ sotto *$\theta$-sussunzione* se $C \leq D$ e, per ogni altra clausola $E$ tale che $E \leq D$, si ha anche $E \leq C$.

Questa definizione si estende immediatamente alla nozione di *lgg* di un insieme di clausole. La clausole $C$ è la generalizzazione meno generale di un insieme $S$ di clausole se $C$ è l' *lgg* di ogni clausola in $S$.

Due clausole $C$ e $D$ sono *equivalenti* sotto *$\theta$-sussunzione* se $C$ *$\theta$-sussume* $D$ e $D$ *$\theta$-sussume* $C$. Una clausola è *ridotta* se non è equivalente sotto *$\theta$-sussunzione* ad alcun sottoinsieme proprio di se stessa.

Ad esempio, se $C = figlia(X,Y) \leftarrow genitore(Y,X)$, che può essere riscritta come insieme di letterali $\{figlia(X,Y), \neg genitore(Y,X)\}$, applicando la sostituzione $\theta = \{X/maria, Y/anna\}$ alla clausola $C$ si ottiene la clausola $C\theta = D = figlia(maria, anna) \leftarrow genitore(anna, maria)$, cioè $C$ *$\theta$-sussume* $D$. La clausola $C$ *$\theta$-sussume* la clausola $D = figlia(X,Y) \leftarrow femmina(X), genitore(Y,X)$ sotto la sostituzione vuota in quanto $\{figlia(X,Y), \neg genitore(Y,X)\}$ è un sottoinsieme proprio di $\{figlia(X,Y), \neg femmina(X), \neg genitore(Y,X)\}$. La clausole $C$ *$\theta$-sussume* anche la clausola $D = figlia(maria, anna) \leftarrow femmina(maria), genitore(anna, maria)$ sotto la sostituzione $\theta = \{X/maria, Y/anna\}$. Infine $C = figlia(X,Y) \leftarrow genitore(Y,X), genitore(W,V)$ e $d = figlia(X,Y) \leftarrow genitore(Y,X)$ sono equivalenti sotto *$\theta$-sussunzione.* La clausola $D$ è ridotta mentre la clausola $C$ non lo è.
Consideriamo ora le seguenti clausole:

$C_1 = ha\_ali(tweety) \leftarrow uccello(tweety)$

$C_2 = ha\_ali(ciccio) \leftarrow uccello(ciccio)$

la clausola $C_3 = ha\_ali(X)$ è chiaramente una generalizzazione di $C_1$ e $C_2$ ma non è la meno generale. $C_3$ si può leggere come "qualsiasi cosa ha le ali" che è chiaramente sbagliata: la sovragenerализzazione può portare a fare errori. Quindi il concetto di generalizzazione minimale è molto importante. La generalizzazione meno generale di $C_1$ e $C_2$ è data da:

$C_4 = ha\_ali(X) \leftarrow uccello(X)$

La nozione di generalità sintattica introdotta è corretta, infatti se $C \leq D$ allora $C \models D$. L'inverso, comunque, non è sempre vero. La relazione di *$\theta$-sussunzione* è più debole della implicazione logica. Consideriamo, ad esempio le due clausole:

$C = pari(X) \leftarrow pari(meta(X))$

$D = pari(X) \leftarrow pari(meta(meta(X)))$

È evidente che $C$ implica $D$ ma $C \not\leq D$ perchè non esiste una sostituzione $\theta$ tale che $C\theta \subseteq D$. La scelta della *$\theta$-sussunzione* è giustificata dal fatto che la *$\theta$-sussunzione* tra due clausole è decidibile e relativamente facile da calcolare sebbene sia $NP$-completa. In generale un *lgg* di due clausole potrebbe non esistere, ma, se due clausole di Horn hanno un simbolo di predicato in comune con lo stesso segno, allora esse hanno un *lgg* sotto *$\theta$-sussunzione.* Viceversa, l'implicazione tra clausole di Horn è indecidibile e non sempre esiste una generalizzazione minima sotto implicazione tra due clausole di Horn.

Vediamo ora come si fa a calcolare la generalizzazione meno generale. Quando si considerano atomi con lo stesso simbolo di predicato e lo stesso segno la generalizzazione meno generale è l'opposto della unificazione. Cioè, dati due termini $f_1$ ed $f_2$ e l'ordinamento imposto da $\leq$, allora l'*lgg* di $f_1$ ed $f_2$ è il loro greatest lower bound mentre l'unificatore più generale è il loro least upper bound. Per calcolare l' *lgg* di due esempi (cioè clausole), abbiamo bisogno di calcolare l' *lgg* di due letterali e quindi l' *lgg* di due termini.
L' *lgg* di due termini $f_1(l_1, \ldots, l_n)$ ed $f_2(l_1, \ldots, m_n)$ è una nuova variabile $v$ se $f_1 \neq f_2$, altrimenti è definito come

$lgg(f_1, f_2) = f_1(lgg(l_1, m_1), \ldots, lgg(l_n, m_n))$.
L' $lgg$ di due letterali $L_1 = (\neg)p(t_1, \ldots, t_n)$ e $L_2 = (\neg)q(s_1, \ldots, s_n)$ è indefinito se $L_1$ ed $L_2$ non hanno lo stesso simbolo di predicato e lo stesso segno, altrimenti è definito come
$lgg(L_1, L_2) = (\neg)p(lgg(t_1, s_1), \ldots, lgg(t_n, s_n))$.
L'$lgg$ di due clausole $C_1 = \{l_1, \ldots, l_k\}$ e $C_2 = \{m_1, \ldots, m_n\}$ è definito come
$lgg(C_1, C_2) = \{lgg(l, m) \mid l \in C_1 \wedge m \in C_2 \wedge lgg(l, m) \text{ è definito}\}$.
Come conseguenza di questa definizione si ha che la lunghezza dell' $lgg$ di due clausole può essere al più $\mid C_1 \mid \times \mid C_2 \mid$.

Il concetto di generalizzazione meno generale è molto importante per la programmazione logica induttiva in quanto forma la base per la definizione di algoritmi di generalizzazione cosiddetti *cauti* che eseguono una ricerca bottom-up nello spazio delle ipotesi. La generalizzazione cauta assume che se due clausole sono vere allora è molto probabile che la loro generalizzazione meno generale sia vera. Comunque può succedere che la generalizzazione meno generale non sia molto utile per alcuni problemi di $ILP$. In genere, quando si vuole generalizzare, è molto più utile farlo rispetto alla conoscenza di dominio. Si parla in tal caso di *generalizzazione più generale relativa (rlgg)*. La generalizzazione meno generale relativa di due atomi ground $A_1$ e $A_2$ (che sono esempi positivi), relativa alla conoscenza di dominio $B$, dove $B$ è ristretta a soli fatti ground, è definita come:
$rlgg(A_1, A_2) = lgg((A_1 \leftarrow K), (A_2 \leftarrow K))$
dove $K$ è la congiunzione di tutti i fatti ground di $B$. Supponiamo di avere un programma logico (cioè la conoscenza sul dominio) costituito dai due soli fatti $A = uccello(tweety)$ e $B = uccello(ciccio)$ e i due esempi positivi $E_1 = ha\_ali(tweety)$ e $E_2 = ha\_ali(ciccio)$. Le due clausole che si possono costruire sono $C_1 = ha\_ali(tweety) \leftarrow uccello(tweety), uccello(ciccio)$, e $C_2 = ha\_ali(ciccio) \leftarrow uccello(tweety), uccello(ciccio)$. Se si costruisce l' $lgg$ di $C_1$ e $C_2$ si ottiene $lgg(C_1, C_2) = ha\_ali(X) \leftarrow uccello(tweety), uccello(ciccio), uccello(X)$. Eliminando i letterali ridondanti si ottiene $ha\_ali(X) \leftarrow uccello(X)$.

**Risoluzione inversa**. La *risoluzione inversa* è un'altra tecnica di generalizzazione che si basa sulla inversione della *regola di risoluzione* dell'inferenza deduttiva. Date due clausole $C_1$ e $C_2$, siano $l_1$ ed $l_2$ due letterali di $C_1$ e $C_2$ rispettivamente. Supponiamo che le due clausole non abbiano variabili con lo stesso nome. Sia $\theta$ l'unificatore più generale ($mgu$) di $l_1$ ed $l_2$ tale che $\neg l_1\theta = l_2\theta$. Poichè $l_1$ ed $l_2$ non hanno variabili in comune questo si può riscrivere $\neg l_1\theta_1 = l_2\theta_2$, dove $\theta = \theta_1\theta_2$. Applicando la regola di risoluzione alle due clausole genitore, si ottiene il risolvente $C = (C_1 - \{l_1\})\theta_1 \cup (C_2 - \{l_2\})\theta_2$. È noto che una clausola $C$ è conseguenza di una teoria $T$ se si può derivare la clausola vuota da $T \wedge \neg C$ mediante ripetute applicazioni della regola di risoluzione. Cosi come la risoluzione forma la base della inferenza deduttiva, la risoluzione inversa può essere usata come base per la inferenza induttiva. Mentre la risoluzione genera la clausola risolvente $C$ a partire dalle due clausole genitore $C_1$ e $C_2$, la risoluzione inversa è in grado di generare uno dei due genitori, dati il risolvente e l'altro genitore. Sono stati definiti due operatori per fare risoluzione inversa che sono *assorbimento* e *identificazione*.

*Assorbimento*

Sia $C_2$ la clausola genitore che contiene il letterale negato. L'operatore di *assorbimento* costruisce la clausola $C_2$, dati il risolvente $C$ e la clausola genitore $C_1$. L'uso del termine assorbimento è chiaro se si tratta di clausole di Horn: il corpo di $C_1$ è assorbito nel corpo di $C$ (dopo opportuna unificazione) e sostituito con la sua testa. Ad esempio, siano:
$C = uccello(tweety) \leftarrow pennuto(tweety), ha\_ali(tweety), ha\_becco(tweety)$.
$C_1 = vola(X) \leftarrow pennuto(X), ha\_ali(X)$.
Il corpo di $C_1$ $\{pennuto(X), ha\_ali(X)\}$ è assorbito nel corpo di $C$ dopo la sostituzione $\theta =$

$\{X/tweety\}$ ed è sostituito con la sua testa $vola(X)\theta$, dando come possibile soluzione :
$C_2 = uccello(tweety) \leftarrow vola(tweety), ha\_becco(tweety).$

*Identificazione*
Sia $C_1$ la clausola genitore che contiene il letterale positivo. L'operatore di *identificazione* costruisce $C_1$ dati il risolvente $C$ e la clausola genitore $C_2$. L'idea è di identificare parte del corpo di $C_2$ nel corpo di $C$ attraverso una sostituzione $\theta$. Supponiamo allora che le teste di $C$ e $C_2$ siano unificabili attraverso $\theta$, e che esattamente un letterale $l$ nel corpo di $C_2$ non ricorre nel corpo di $C$. L' identificazione costruisce $C_1$ usando il letterale $l$ come testa e come corpo la parte del corpo di $C$ che non c'è in $C_2$. Ad esempio, per le due clausole $C$ e $C_2$ di sopra $l = vola(tweety)$, $C - C_2 = \{pennuto(tweety), ha\_ali(tweety)\}$. Quindi abbiamo:
$C_1 = vola(tweety) \leftarrow pennuto(tweety), ha\_ali(tweety).$

### 6.2.2 Metodi Top-down

I metodi top-down di programmazione logica induttiva imparano programmi generando clausole per mezzo della specializzazione. In generale un programma $P$ è arricchito con nuove clausole finchè esiste qualche esempio positivo non ancora coperto dal programma. La generazione di una clausola $C$ avviene considerando prima la clausola con corpo vuoto, quindi la clausola è specializzata aggiungendo ai suoi antecedenti un letterale finchè la clausola non copre nessun esempio negativo. Se il nuovo letterale aggiunto fa si che la clausola non copra nemmeno un esempio positivo allora si deve procedere alla scelta di un nuovo letterale. Una procedura molto generale che realizza i metodi top-down è la seguente:

**while** qualche esempio positivo non è coperto da una clausola $C \in P$ **do**
  Genera una clausola $C = P(\overline{X}) \leftarrow \alpha$, $\alpha$ essendo una congiunzione di letterali
  Rimuovi gli esempi positivi ricoperti da $C$
  Aggiungi $C$ a $P$

La generazione di una clausola avviene come segue:

Seleziona un predicato $P$ e si ponga $C = P(\overline{X}) \leftarrow true$
**while** $C$ copre esempi negativi **do**
  Seleziona un letterale $L$
  Aggiungi $L$ agli antecedenti di $C$, $\alpha = \alpha \wedge L$
  **if** $C$ non copre alcun esempio positivo **then** scegli un' altro letterale
**return** $C$ (oppure fallisci se tutte le scelte per $L$ sono state considerate)

Una caratteristica dei sistemi top-down consiste nel fatto che essi usano un *criterio estensionale* per verificare se una clausola $C = P(\overline{X}) \leftarrow \alpha$ copre un esempio $P(\overline{a})$. Questo criterio considera un esempio $P(\overline{a})$ coperto da una clausola se è possibile derivare questo esempio dalla conoscenza di dominio e dagli esempi positivi, indipendentemente dalle altre clausole che saranno generate nel programma finale. Ad esempio, se abbiamo:
$E^+$ :
$e_1 = append([a,b],[c,d],[a,b,c,d])$
$e_2 = append([b],[c,d],[b,c,d])$
$e_3 = tail([a,b],[b])$
conoscenza di dominio $BK$:
$head([A \mid \_], A)$
$cons(C, D, [C \mid D])$
allora la clausola :

$C = append(X,Y,Z) \leftarrow head(X,H), tail(X,T), append(T,Y,W), cons(H,W,Z)$
copre estensionalmente $e_1$ perchè $BK$ deriva $head([a,b],H)$ con $H = a$, se $T = [b]$, allora $tail([a,b],[b])$ è un esempio positivo di $tail$; $append([b],[c,d],[b,c,d])$ è un esempio positivo di $append$ e $BK$ deriva $cons(a,[b,c,d],Z)$ con $Z = [a,b,c,d]$.

Il ricoprimento estensionale, quindi, nella valutazione di una clausola richiede solo la clausola da valutare, gli esempi positivi e la conoscenza di dominio, senza dover condiderare le altre clausole che saranno generate nel programma finale. Un vantaggio di questo criterio è che la generazione di una clausola non influenza la valutazione delle altre. Uno svantaggio, invece, è il fatto che l'utilizzo del programma finale che si è generato valuta gli antecedenti delle clausole non guardando solo gli esempi positivi, ma con delle chiamate alle clausole corrispondenti. Questo ha come conseguenza che il comportamento del programma a tempo di esecuzione differisce dal comportamento del programma stesso durante la sua generazione. Quindi si potrebbero avere dei risultati inaspettati.

Uno dei primi sistemi $ILP$ top-down sviluppati è stato il *Model Inference System (MIS)* di Shapiro [38]. Lo scopo di $MIS$ è di inferire una assiomatizzazione, attraverso clausole di Horn, di un modello sconosciuto in base a dei fatti ground che sono veri e falsi in $M$. Il metodo è incrementale, cioè l'ipotesi induttiva corrente (un programma logico $P$) è modificata se vengono forniti nuovi esempi.

Un altro importante sistema di apprendimento top-down è $FOIL$ di Quinlan [33, 34]. La sua implementazione efficiente lo ha reso come il metodo base con cui tutti gli altri sistemi sviluppati si devono confrontare. $FOIL$ è un metodo puramente top-down, la sola differenza, che ne costituisce la sua caratteristica principale, è che nella generazione di una nuova clausola non viene fatto nessun backtracking: una volta che un letterale è stato scelto per essere aggiunto nella parte antecedente della clausola non vengono prese in considerazione altre alternative. La scelta del letterale, però, è fatta con una tecnica euristica che si basa sul numero di esempi positivi ricoperti e negativi non ricoperti. In pratica si considerano tutte le possibili scelte di letterali e, per ogni letterale, tutte le possibili configurazioni del letterale rispetto alle variabili e si prende il letterale che fornisce il guadagno maggiore, dove per guadagno si intende che copre il massimo numero di esempi positivi ed il minimo numero di esempi negativi. Uno dei principali problemi di questa euristica è che a volte la scelta di un letterale può non dare alcun guadagno mentre la sua scelta accoppiata ad un altro letterale potrebbe essere necessaria per costruire il programma corretto. Per evitare questo problema sono state proposte varie tecniche che permettono di "guardare in avanti" per più di un predicato, in base alla conoscenza a disposizione.

## 6.3 Reti Neurali

Le reti neurali artificiali (generalmente dette reti neurali) presentano una struttura ed un modello di computazione molto differenti dai modelli di elaborazione definiti nel campo dell'intelligenza artificiale. L'idea che sta alla base della ricerca sulle reti neurali è che si possono ottenere sistemi intelligenti usando uno schema computazionale in qualche modo simile a quello del cervello umano. Per simulare il modello del cervello umano, le reti neurali sono composte da un numero elevato di neuroni (artificiali) che comunicano tra loro attraverso un insieme di connessioni che svolgono il ruolo delle sinapsi. Occorre comunque precisare che le reti neurali non rappresentano che un modello molto semplificato del cervello. Nelle reti neurali, diversamente da quasi tutti i modelli computazionali, non esiste un programma che specifica le operazioni da eseguire, ma la computazione è definita attraverso le caratteristiche delle unità di elaborazione (*neuroni artificiali*) e delle loro interconnessioni [26, 17]. Una rete neurale apprende attraverso l'esperienza piuttosto che attraverso un programma che specifica il suo comportamento. Questa è la fondamentale differenza rispetto all'altro approccio dell'Intelligenza Artificiale (IA) basato sulla manipolazione di simboli attraverso l'uso di regole che costituiscono il programma da eseguire.

Le caratteristiche principali di una rete neurale sono:

- un elevato numero di semplici unità di elaborazione (*neuroni artificiali*),
- un elevato numero di connessioni tra i neuroni (*sinapsi*),
- uno schema di controllo parallelo e distribuito,
- un algoritmo di apprendimento.

L'obiettivo di una rete neurale è quello di associare un output $y$ ad un insieme di segnali di input $x_1$, $x_2$, ..., $x_n$. In una rete neurale non esiste un controllore globale dello stato della computazione. Ogni neurone artificiale calcola il proprio stato $y$ sulla base di informazioni locali, cioè evolve secondo una propria legge specificata tramite una funzione di transizione $f$ che dà la dipendenza dello stato $y$ dai segnali ricevuti dalle connessioni. L'insieme degli stati dei neuroni della rete costituisce lo stato globale della rete. In generale $f$ è una funzione nonlineare della combinazione lineare degli input tramite dei coefficienti che sono dati dai pesi $(p_{ij})$associati alle connessioni del neurone:

$$y_i = f(\sum_{j=1}^{n} p_{ij} x_j)$$

Come accennato in precedenza, una tra le più interessanti caratteristiche delle reti neurali consiste nella capacità di apprendimento dall'esperienza, diversamente dai sistemi dell'IA classica dove la conoscenza del sistema deve essere esplicitamente programmata [10]. Molti algoritmi di apprendimento (*learning algorithms*) presentano la caratteristica di rinforzare le connessioni fra elementi che hanno gli stessi valori di attivazione nello stato coerente che deve essere memorizzato. In particolare, le regole di apprendimento specificano un insieme iniziale di pesi associati alle connessioni e indicano come questi pesi devono essere adattati per migliorare le prestazioni della rete [39] [44]. L'apprendimento può avvenire, almeno parzialmente, presentando alla rete neurale una serie di esempi che costituiscono il cosiddetto *training set*. Solo dopo che la rete ha imparato a rispondere correttamente a tutti gli input del training set, essa è pronta per essere utilizzata per risolvere i problemi per cui è stata progettata.

I modelli di reti neurali finora proposti possono essere divisi in quattro categorie [47] :

- $Auto-associators$, che associano i pattern memorizzati a loro stessi.
- $Pair-associators$, che associano i pattern memorizzati ad altri pattern.
- *Supervised learning*, in cui la fase di apprendimento guidata da un istruttore che corregge gli errori della rete.
- *Unsupervised learning*, in cui non esiste un istruttore, la rete deve imparare da sola a classificare i pattern di input.

Dai primi studi di McCulloch e Pitts, molti modelli di reti neurali sono stati proposti [23, 46]. Alcuni modelli di reti neurali tra i più noti sono: le Reti di Hopfield, il Percettrone, le

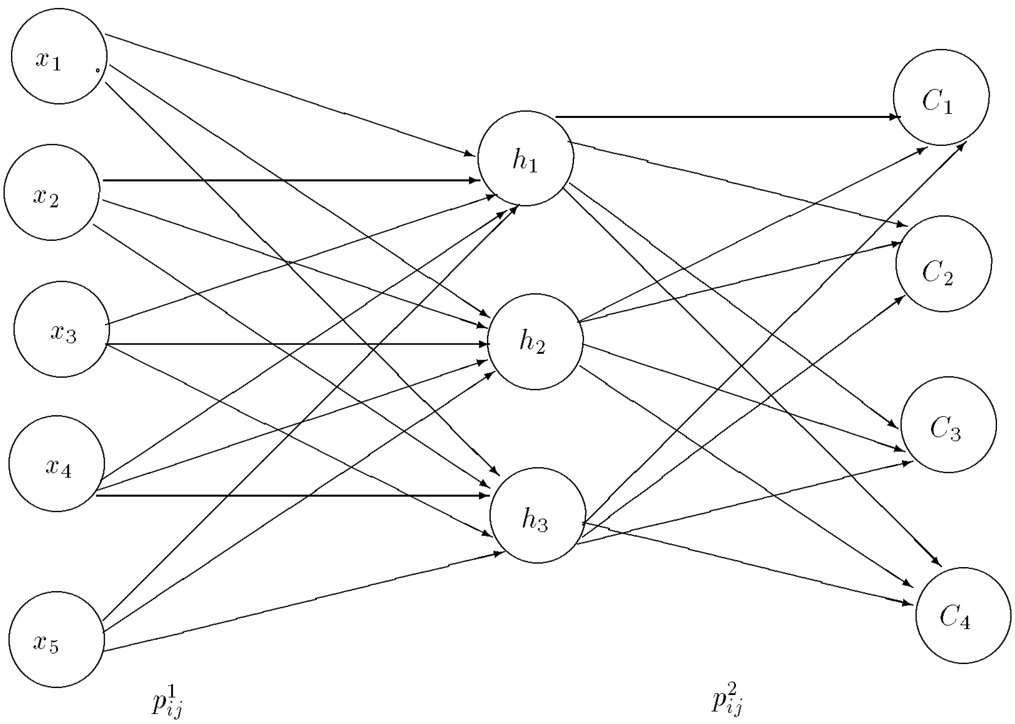



Figure 2: Esempio di rete neurale Multilayer per la classificazione di dati.

Reti Multilayer, la Macchina di Boltzmann, le Self-Organizing Map e le Adaptive Resonance Networks.

Negli ultimi anni alcuni di questi modelli di reti neurali sono stati usati per la realizzazione di applicazioni di data mining [3]. Esempi di sistemi basati su reti neurali che vengono usati per realizzare applicazioni di data mining sono Neural Connection, MATLAB Neural Network Toolbox, IBM Neural Network Utility, BackPack, NeuroSolutions, BrainMaker e ModelQuest.

Come detto in precedenza, una rete neurale apprende dall'esperienza e raggiunge una decisione in base ai dati che ad essa vengono forniti. Allo stesso modo come le persone imparano dalla loro esperienza ed in base alla conoscenza che hanno prendono delle decisioni. Infatti, le reti neurali ricevono in input un insieme di dati che possono contenere dell'informazione "sporca" e cercano di arrivare a delle conclusioni (output) in base a quanto da esse appreso nel passato e che è codificato nei valori dei neuroni e dei pesi associati alle loro connessioni. Quindi, le reti neurali possedendo la capacità di riconoscere pattern da insiemi di dati di input possono effettuare una serie di operazioni tipiche del data mining come la classificazione di un insieme di dati, il clustering, l'associazione di due o più informazioni, e l'analisi dei legami tra dati.

Come esempio descriviamo brevemente come una rete neurale Multilayer permette di realizzare una funzione di classificazione. Questo modello di rete neurale è composta da più strati di neuroni tali che l'output di un neurone in uno strato è connesso con l'input di tutti i neuroni dello strato successivo (vedi figura 2). Gli input del primo strato costituiscono l'input della rete neurale e gli output dell'ultimo strato costituiscono l'output della rete. Una rete multi-layer è adatta all'implementazione di una funzione di classificazione che assegna un oggetto (dato) definito da un vettore di ingresso $[x_1, x_2, ..., x_n]$ ad una o più classi $C_1, C_2, ..., C_m$ che costituiscono i neuroni dell'output della rete. Se l'oggetto appartiene alla classe $C_i$, il neurone di output $C_i$ ha un valore 1, altrimenti, se l'oggetto non appartiene alla classe $C_i$ il neurone di output $C_i$ ha un valore 0. In figura 2 è mostrato un esempio di rete neurale con $n = 5$ ed $m = 4$. In questo modo, selezionando in maniera appropriata l'insieme dei pesi per tutte le connessioni della rete neurale, essa potrà realizzare un insieme di funzioni di classificazione. La scelta corretta dei pesi potrà essere fatta fornendo alla rete una serie di esempi che sono rappresentati dai valori di input e i corrispondenti valori di output (supervised learning). Secondo questo approccio, un esempio è costituito da un vettore di valori di input con il desiderato valore di output dove l'insieme dei $C_i$ hanno i valori 1 o 0 in dipendenza dell'appartenenza o meno dell'input alla classe $i$. La rete aggiusterà i pesi delle connessioni in base agli esempi forniti ed al termine della procedura di apprendimento sarà pronta a classificare oggetti nuovi e differenti da quelli del training set.

In base a queste caratteristiche le reti neurali rappresentano un naturale ed utile modello per effettuare il data mining in grandi database. Oltre a ciò, occorre sottolineare che per la loro struttura, le reti neurali si prestano molto bene ad essere implementati sui calcolatori paralleli. Quest'ultima caratteristica permette di utilizzare le reti neurali per realizzare applicazioni di data mining ad alte prestazioni [42].

## 6.4 Algoritmi Genetici

Mentre le reti neurali sono state proposte come un modello semplificato del cervello umano, gli algoritmi genetici rappresentano un modello dei meccanismi biologici di selezione naturale nelle specie. Infatti, gli algoritmi genetici sono algoritmi stocastici di ricerca basati sulla meccanica della selezione naturale e sul paragone con la genetica osservata in natura. Essi combinano la sopravvivenza degli elementi più forti di una popolazione con lo scambio di conoscenza strutturata per formare un algoritmo di ricerca innovativo. In ogni generazione, un nuovo insieme di creature artificiali (stringhe) viene creato usando sottostringhe dei migliori elementi della generazione precedente. Occasionalmente vengono provate parti nuove per introdurre possibili miglioramenti.

Gli algoritmi genetici sono stati introdotti da John Holland [19]. L'obiettivo della ricerca di Holland era duplice: astrarre e spiegare in modo rigoroso il processo di adattamento dei sistemi naturali e progettare sistemi software che mantengano il meccanismo naturale. Questo approccio ha portato a scoperte importanti nelle scienze naturali e dei sistemi artificiali. Nonostante gli algoritmi genetici siano basati sulla casualità, non sono una semplice ricerca casuale, infatti essi esplorano efficacemente informazioni storiche per dedurre nuovi punti di ricerca che abbiano migliori caratteristiche. Essi appartengono alla categoria degli algoritmi probabilistici, ma sono molto differenti dagli algoritmi casuali, perchè combinano elementi di ricerca stocastica e ricerca guidata.

Il tema centrale della ricerca negli algoritmi genetici è stata la robustezza, cioè il giusto equilibrio tra efficienza ed efficacia, necessario per sopravvivere in molti ambienti diversi. Le implicazioni della robustezza nei sistemi artificiali sono molteplici: se un sistema artificiale può essere reso più robusto, eventuali e costose riprogettazioni possono essere ridotte o del tutto eliminate. Se si raggiungono elevati livelli di adattamento, i sistemi possono svolgere le loro funzioni più a lungo ed in maniera migliore. I progettisti di sistemi artificiali, sia software che hardware, possono essere colpiti positivamente davanti alla robustezza, l'efficienza e la flessibilità dei sistemi biologici: caratteristiche come il miglioramento, la riproduzione e la cura dei difetti sono la regola nei sistemi naturali, mentre compaiono appena, in forma molto semplice, solo nei più sofisticati sistemi artificiali.

Dopo essersi affermati come un valido approccio per problemi che richiedono una ricerca efficiente, gli algoritmi genetici stanno trovando applicazione in campo ingegneristico, scientifico, economico. Le principali ragioni che stanno dietro a questo successo sono chiare: questi algoritmi sono computazionalmente semplici, ma potenti nella loro ricerca del miglioramento, ed inoltre non sono limitati da assunzioni restrittive riguardo allo spazio di ricerca, come ad esempio, continuità e derivabilità.

Come prima accennato, gli algoritmi genetici eseguono una ricerca multidirezionale mantenendo una popolazione di soluzioni potenziali ed incoraggiando la formazione di blocchi di informazione ed il loro interscambio tra le soluzioni. La popolazione passa attraverso una evoluzione simulata: ad ogni generazione le soluzioni relativamente "buone" si riproducono, mentre quelle relativamente "cattive" muoiono. Per distinguere la bontà delle soluzioni viene usata una funzione di *fitness* che esegue la valutazione delle soluzioni ottenute come possibili soluzioni del problema.

In un algoritmo genetico un *gene* viene solitamente rappresentato solitamente con un bit, un carattere, o un intero, a seconda della cardinalità dell'insieme dei valori che il gene può assumere. L'insieme dei geni di un individuo costituiscono il *cromosoma*. Il cromosoma, è solitamente rappresentato con un array di geni, come definiti precedentemente. A sua volta la popolazione viene anch'essa rappresentata come un array o comunque una collezione di oggetti di tipo individuo. La struttura di algoritmo genetico può essere rappresentata come composta dai seguenti passi:

1. $t = t + 1$ ;
2. Selezione di $P(t)$ da $P(t-1)$ ;
3. Ricombinazione di $P(t)$ ;
4. Valutazione di $P(t)$ ;

dove $t$ rappresenta l'orologio evolutivo, e $P(t)$ la popolazione al tempo $t$. Queste operazioni vengono ripetute fino al raggiungimento di una soluzione soddisfacente. Ad ogni iterazione $t$, un algoritmo genetico mantiene una popolazione di potenziali soluzioni, $P(t) = \{s_1^t, ..., s_n^t\}$. Ogni soluzione $s_i^t$ viene valutata per misurare la sua fitness, cioè la sua bontà nell'ambiente di appartenenza. Successivamente una nuova popolazione (iterazione $t+1$) viene costruita selezionando gli individui migliori. Alcuni individui di questa nuova popolazione vengono sottoposti a *riproduzione* e *mutazione*, per formare nuove soluzioni.

La riproduzione, detta *crossover*, combina le caratteristiche di due genitori per formare una coppia di figli, scambiando segmenti corrispondenti dei genitori.

Per esempio, se i genitori sono rappresentati con i vettori $(a_1, b_1, c_1, d_1, e_1, f_1)$ e $(a_2, b_2, c_2, d_2, e_2, f_2)$, allora incrociando i cromosomi dopo il secondo gene, si possono produrre come figli $(a_2, b_2, c_1, d_1, e_1, f_1)$ e $(a_1, b_1, c_2, d_2, e_2, f_2)$. La giustificazione intuitiva del crossover, è lo scambio di informazioni tra differenti potenziali soluzioni. Un esempio di crossover è riportato in figura 3. La mutazione altera arbitrariamente uno o più geni del cromosoma selezionato, usando un cambiamento casuale con probabilità prestabilita (vedi figura 4). La giustificazione intuitiva della mutazione, è l'introduzione di variabilità nella popolazione, per introdurre soluzioni che altrimenti non verrebbero esplorate dal solo crossover.

Un algoritmo genetico applicato ad un problema specifico, deve avere queste componenti:

1. una rappresentazione genetica per le potenziali soluzioni del problema,
2. un modo per creare una popolazione iniziale di possibili soluzioni,

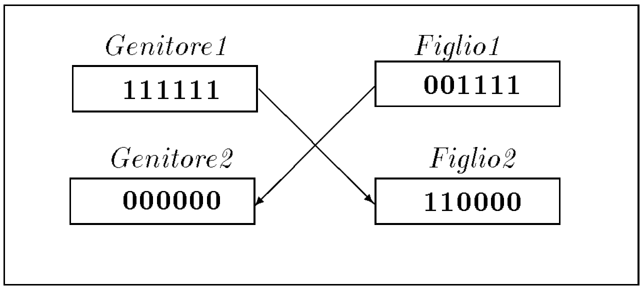



Figure 3: Esempio di crossover.

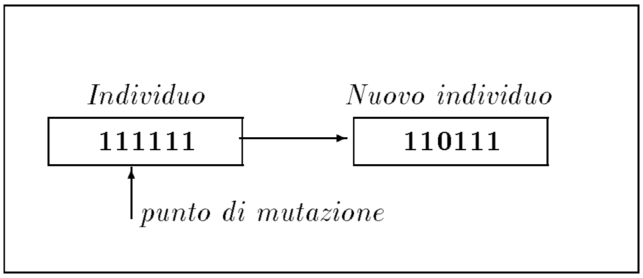



Figure 4: Esempio di mutazione.

3. una funzione di valutazione, che giochi il ruolo dell'ambiente valutando la bontà delle soluzioni trovate,

4. operatori genetici per creare e mutare i figli durante la fase di ricombinazione,

5. valori per i vari parametri dell'algoritmo (dimensione della popolazione, probabilità di applicazione degli operatori genetici).

Le attività di ricerca svolte nell'area delle tecniche di machine learning tramite l'uso di algoritmi genetici possono essere raggruppati secondo due approcci principali.

- Approccio *Pittsburgh* [40] : ogni individuo o entità della popolazione è un insieme di regole che rappresentano una completa soluzione al problema (*disgiunzione delle descrizioni*); l'intera popolazione rappresenta un insieme di descrizioni complete che sono in competizione tra loro. La dimensione di ogni individuo è quindi un multiplo della lunghezza di ogni singola descrizione. Per eseguire l'operazione di crossover ambedue le stringhe che corrispondono agli individui genitori sono tagliati nello stesso punto modulo la lunghezza della disgiunzione delle descrizioni. Operando in questo modo gli individui figli che si ottengono rappresentano a loro volta delle soluzioni complete.

- Approccio *Michigan* [20] : ogni elemento della popolazione consiste di regole individuali che rappresentano una singola soluzione parziale del problema (*descrizione*) ; l'intera popolazione forma una singola regola di classificazione, cioè una regola la cui descrizione consiste della disgiunzione di tutte le descrizioni rappresentate dagli individui della popolazione. In questo caso l'operazione di crossover si realizza spezzando le stringe dei due individui genitori nella stessa posizione e creando due nuove stringhe combinando la prima parte della stringa di un genitore con la seconda parte della stringa dell'altro genitore e viceversa. La bontà di un individuo si misura in base alla sua accuratezza di classificazione (rapporto esempi positivi/esempi negativi coperti) e alla sua generalità (numero di esempi coperti rispetto al numero totale di esempi coperti da tutta la popolazione).

Uno dei primi sistemi di machine learning basato sugli algoritmi genetici è LS-1 [41] che ha introdotto una rappresentazione strutturata basata sulla semantica del dominio in cui ricade il problema da risolvere tramite operatori genetici. Un altro sistema che come LS-1 è basato sull'approccio Pittsburgh è GABIL, in cui le descrizioni dei concetti, sono definite tramite una collezione di regole di classificazione che si possono sovrapporre. Un altro sistema basato sugli algortimi genetici è OMEGA Predictive Modelling System. Un sistema sviluppato recentemente è GA-MINER [16]. GA-MINER è uno strumento di ricerca basato sugli algoritmi genetici che supporta diversi modelli di pattern, come pattern di correlazione, pattern con regole esplicite e *distribution shift patterns*, e lavora con differenti livelli di supervisione da parte dell'utente. L'algoritmo genetico usato in GA-MINER usa un modello di popolazione strutturata in cui ogni partner riproduttivo di un genoma è selezionato nell'ambito del suo vicinato. Gli operatori di crossover e mutazione sono definiti a differenti livelli sulla base della struttura della rappresentazione. L'evoluzione avviene sostituendo la regola che si adatta peggio con la regola che si adatta meglio.

# 7 Sistemi di Data Mining

In questa sezione sono presentati alcuni esempi di sistemi di data mining. Lo scopo è quello di presentare, per ognuno dei metodi di rappresentazione prima descritti, un esempio di sistema significativo.

## 7.1 ID3

Il sistema $ID3$ (*Induzione di Alberi di Decisione*) è stato sviluppato agli inizi degli anni ottanta da Quinlan [31] ed ha avuto un notevole impatto sulla ricerca in machine learning. $ID3$ è un sistema di supervised learning che costruisce alberi di decisione da un insieme di esempi. Questi esempi sono tuple per cui il dominio di ogni attributo è limitato ad un insieme piccolo di valori. La conoscenza è rappresentata attraverso un albero di decisione, pertanto la spazio di ricerca consiste di tutti gli alberi che possono essere costruiti con gli attributi ed i valori dell'insieme degli esempi. Tra tutti gli alberi nello spazio di ricerca, il sistema deve trovare il migliore. Il sistema usa una strategia di ricerca top-down senza backtracking. L'albero di ricerca è costruito come segue:

1. si seleziona un attributo come radice dell'albero e si generano tanti rami quanti sono i possibili valori di quell'attributo;

2. l'albero è usato per classificare il training set. Se tutti gli esempi di una particolare foglia appartengono alla stessa classe, questa foglia è etichettata con quella classe. Se tutte le foglie sono etichettate con una classe l'algoritmo termina;

3. altrimenti il nodo è etichettato con un attributo che non ricorre sul cammino dalla radice al nodo e si creano i rami per tutti i possibili valori. Si ricomincia, quindi, dal passo 2.

Questo algoritmo crea un albero che classifica correttamente tutti gli oggetti del training set ma l'albero generato non è il più semplice. La scelta dell'attributo da espandere si basa su una funzione euristica che valuta la bontà di un attributo in base al concetto di *guadagno informativo massimo*, cioè si seleziona l'attributo che minimizza l'informazione necessaria per classificare gli esempi nel sottoalbero risultante. Le funzioni di valutazione più usate per sviluppare un nodo lavorano sulla riduzione del grado di casualità o impurità nel nodo corrente. Una delle funzioni di valutazione più usate è la funzione *entropia* definita come

$$-\sum_{j} p_j \; log_2 \; p_j$$

dove $p_j$ è la probabilità della classe $j$. In particolare, se si indicano con $P$ la classe contenente tutti gli esempi positivi, $N$ la classe contenente tutti gli esempi negativi ed $S$ un insieme di esempi contente $p$ elementi nella classe $P$ ed $n$ elementi della classe $N$, si definisce la misura della quantità di informazione necessaria per decidere se un esempio arbitrario appartiene a $P$ o $N$ la quantità

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Si supponga che un attributo $A$ sia la radice dell'albero e che esso suddivida l'insieme $S$ negli insiemi $\{S_1, \ldots, S_h\}$. Se $S_i$ contiene $p_i$ esempi positivi ed $n_i$ esempi negativi, l'informazione necessaria per decidere se un elemento in $S_i$ appartiene a $P$ o ad $N$ è $I(p_i, n_i)$. Quindi, l'informazione necessaria per classificare un elemento di $S$ usando un albero con l'attributo $A$ come radice, è la media pesata dell'informazione necessaria per classificare gli oggetti in tutti i sottoalberi $S_i$:

$$E(A) = \sum_{i=1}^{h} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

L'attributo $A$ selezionato è quello per cui il guadagno informativo sia massimo, cioè tale che $E(A)$ sia minimo.

## 7.2 AQ15

Il sistema $AQ15$ è stato sviluppato da Michalski [27] ed è un sistema di learning induttivo che genera regole di decisione dove la parte condizionale è una formula logica. Una caratteristica particolare di questo sistema è l'*induzione costruttiva*, cioè l'uso di conoscenza di dominio per generare nuovi attributi che non sono presenti nei dati di input. Ogni regola di decisione definisce una classe. Di conseguenza la regola serve per classificare tutti gli esempi positivi di quella classe e nessun esempio negativo di quella classe. Gli esempi nel training set sono vettori di valori di attributi.

Le regole sono rappresentate nella notazione $VL_1$ (*Variable-valued Logic system 1*). Un *selettore* è una condizione sul valore di un attributo che lega un attributo ad un valore oppure ad una disgiunzione di valori, ad esempi $color = red \vee green$. Una congiunzione di selettori è chiamata un *complesso* (cioè una set-description). La parte condizionale di una regola di decisione è composta da una disgiunzione di complessi ed è chiamata *ricoprimento*. Quando costruisce una regola di decisione, $AQ15$ esegue una ricerca euristica attraverso lo spazio di tutte le possibili espressioni per determinare quella che spiega tutti gli esempi positivi e nessun esempio negativo. Poichè vi sono molte di queste regole, il compito di $AQ15$ è di trovare la più preferita. Questo criterio di preferenza è definito dall'utente per riflettere le necessità del dominio applicativo. Le operazioni che si eseguono sulle regole sono l'aggiunta di un complesso al ricoprimento di una regola e l'intersezione di un insieme di complessi con un insieme di selettori, dove per intersezione si intende l'insieme di tutte le combinazioni di congiunzioni di elementi da entrambi gli insiemi.

Il sistema genera una regola di decisione per ogni classe usando un algoritmo di ricerca irrevocabile. Ad ogni passo si aggiunge al ricoprimento il complesso migliore. Ogni passo incomincia focalizzando l'attenzione su un esempio positivo selezionato, chiamato *seme*. L'algoritmo genera un insieme di tutti i complessi (*stella*) che coprono il seme e non coprono alcun esempio negativo, e quindi selezione il complesso migliore dalla stella in base al criterio definito dall'utente. Questo complesso è aggiunto al ricoprimento. L'algoritmo di ricoprimento base è il seguente:

**while** un ricoprimento parziale non copre tutti gli esempi positivi **do**

1. seleziona un seme, cioè un esempio positivo non ricoperto;
2. genera una stella, cioè determina complessi il più generale possibile che coprano il seme e nessun esempio negativo;
3. seleziona il complesso migliore, come definito dalla funzione di qualità;
4. aggiungi il complesso al ricoprimento.

Il passo 2 che genera una stella è il seguente:

**while** stella parziale copre esempi negativi **do**

1. seleziona un esempio negativo coperto;
2. genera una stella parziale: tutti i selettori generali massimali che coprono il seme ed escludono l' esempio negativo;
3. genera una nuova stella parziale intersecando la stella parziale corrente con la stella parziale costruita finora;
4. aggiusta la stella parziale; cioè mantieni solo i *maxstar* migliori complessi.

$Maxstar$ è un parametro che controlla quanti complessi devono essere mantenuti nella stella parziale. Se infatti la procedura di generazione della stella lavorasse esaustivamente, lo spazio di ricerca dei ricoprimenti crescerebbe molto rapidamente. In tal modo, invece, si mantengono soltanto i complessi migliori, in base al criterio specificato. Per una data stella parziale, la procedura calcola la qualità dei complessi nell'intersezione, ne mantiene $maxstar$ di essi e interseca di nuovo questi ultimi, finchè nessun esempio negativo è ricoperto.

## 7.3 MIS

Uno dei primi sistemi $ILP$ top-down sviluppati è stato il *Model Inference System (MIS)* di Shapiro [38]. Lo scopo di $MIS$ è di inferire una assiomatizzazione, attraverso clausole di Horn, di un modello sconosciuto in base a dei fatti ground che sono veri e falsi in $M$. Il metodo è incrementale, cioè l'ipotesi induttiva corrente (un programma logico $P$) è modificata se vengono forniti nuovi esempi. Il programma ipotizzato può essere modificato per due motivi: (1) $P \vdash e$ ma $e$ è falso in $M$; (2) $P \nvdash e$ ma $e$ è vero in $M$. Nel primo caso $P$ deve essere indebolito, nel secondo caso rafforzato. $P$ può essere indebolito o rimuovendo una delle sue clausole oppure rendendo una clausola più debole, cioè aggiungendo letterali al suo corpo. $P$ può essere rafforzato o aggiungendo una nuova clausola oppure oppure rendendo una delle sue clausole più generali rimuovendo dei letterali dal suo corpo. $MIS$ esegue queste modifiche finchè non ottiene un programma completo e consistente; quindi legge un nuovo esempio. Lo schema algoritmico di base è il seguente:

$P \leftarrow$ il programma che contiene solo la clausola vuota
**repeat**
leggi un nuovo esempio
**while** $P$ non è consistente con gli esempi disponibili **do**
  **if** $P \vdash e$ ed $e$ è un esempio negativo **then**
    *rendi P più debole*
  **if** $P \nvdash e$ ed $e$ è un esempio positivo **then**
    *rendi P più forte*
**forever**
**Exception:** se verificare $P \vdash e$ richiede un numero di passi di risoluzione maggiore di $h(e)$ chiama una procedura di rimozione di una clausola da $P$.

La funzione $h$ restituisce un limite superiore del costo computazionale richiesto quando si esegue un programma $P$ su un dato goal. Più precisamente, un programma $P$ si dice *h-facile* quando:

1. se $e$ è vero in $M$, $P \vdash e$ e la profondità dell'albero di risoluzione corrispondente è al più $h(e)$ e

2. se $e$ è falso in $M$, $P \nvdash e$ con un albero di dimostrazione finitamente fallito di profondità massima $h(e)$.

Un modello $M$ è detto *h-facile* se può essere assiomatizzata da un programma *h-facile*.

Rendere $P$ più forte o più debole richiede una ricerca nello spazio dei possibili programmi logici. $MIS$ evita questo richiedendo all'utente il valore di verità di particolari atomi.

## 7.4 FOIL

Il sistema $FOIL$ (*First Order Inductive Learner*) [31] costruisce clausole di Horn da relazioni. $FOIL$ riceve informazione su una o più relazioni, una delle quali (detta relazione target), deve essere definita attraverso un programma Horn. Per ogni relazione viene dato un insieme di tuple (esempi positivi). Per la relazione target viene dato anche un insieme di tuple che non appartengono alla relazione (esempi negativi). Alternativamente si può assumere la *closed world assumption* per affermare che nessuna tupla, oltre quelle definite, appartiene alla relazione. Il compito di apprendimento consiste nel trovare un insieme di clausole per la relazione target che derivi tutti gli esempi positivi e non copra nessuno degli esempi negativi. In situazioni reali, comunque, la presenza di dati inesatti (noise) previene dall'apprendere delle definizioni complete. Per superare questo problema, $FOIL$ usa una euristica che limita la complessità delle clausole e dei programmi. Le clausole finali potrebbero coprire molti (anche se non tutti) degli esempi positivi e alcuni, piuttosto che nessuno, esempi negativi. Per costruire una definizione di una relazione $Q(V_1, \ldots, V_k)$, $FOIL$ esegue i seguenti passi:

- inizializza l'insieme di addestramento globale con le tuple della relazione target e
- while l'insieme di addestramento contiene tuple positive
  - trova una clausola che descrive parte della relazione $Q$ e
  - rimuovi le tuple positive coperte da questa clausola

Per costruire una clausola $FOIL$ incomincia definendo la testa della clausola con la relazione target. Quindi aggiunge letterali alla parte destra finchè nessuna tupla negativa è ricoperta oppure quando l'euristica decide che la clausola è troppo complessa.

- inizializza la clausola a $Q(V_1, \ldots, V_k) \leftarrow$ e sia $T$ un training set locale costituito dal training set globale da cui sono state eliminate le tuple già coperte da altre clausole;
- while $Q$ copre tuple negative e non è troppo complesso
  - trova un letterale $L$ da aggiungere alla parte destra della clausola;
  - genera un nuovo training set $T'$ ottenuto estendendo le tuple di $T$ come specificato sotto;
  - sostituisci $T$ con $T'$

La scelta di un letterale avviene come segue. Consideriamo una clausola parzialmente sviluppata $A \leftarrow L_1, \ldots, L_{m-1}$ contenente le variabili $V_1, \ldots, V_x$. Ogni tupla del training set è del tipo $(c_1, \ldots, c_x)$ e rappresenta una istanza ground delle variabili presenti nella clausola. Quando si aggiunge un letterale $L_m$ della forma $P(V_{i_1}, \ldots, V_{i_p})$ alla parte destra della clausola, si ottiene una nuova clausola. Se il letterale contiene una o più nuove variabili, l'arità del nuovo training set aumenterà. $x'$ denoti il numero di variabili della nuova clausola. Allora, ogni tupla del nuovo training set $T'$ avrà la forma $(d_1, \ldots, d_{x'})$ con $d_i$ costanti, e avrà le seguenti proprietà: 1) $(d_1, \ldots, d_x)$ è una tupla di $T$ e 2) $(d_{i_1}, \ldots, d_{i_p})$ è nella relazione $P$. Cioè, ogni tupla in $T'$ è una estensione di una tupla in $T$ e, l'istanza ground che essa rappresenta soddisfa il letterale. Ogni tupla in $T$ genera zero o più tuple in $T'$ etichettate positive o negative analogamente alle tuple di $T$ che le hanno generate. Il criterio di scelta di un letterale piuttosto che un altro si basa sul guadagno di informazione. Siano $T_+$ e $T'_+$ il numero di tuple positive in $T$ e $T'$ rispettivamente. La conoscenza che una certa tupla in $T$ sia un esempio positivo parta un contributo informativo pari a

$$I(T) = -log_2(T_+ / \mid T \mid)$$

e analogamente per $I(T')$. Se $I(T')$ è inferiore a $I(T)$ allora l'aggiunta di $L_m$ ci fa *guadagnare* informazione e, se $s$ delle tuple di $T$ hanno estensioni in $T'$, l'informazione totale guadagnata sulle tuple positive è data da

$$gain(L_m) = s \times (I(T) - I(T'))$$

*FOIL* esplora lo spazio di ricerca dei possibili letterali che potrebbero essere aggiunti ad una clausola ad ogni passo cercando quello che produce il massimo guadagno positivo. In tal modo larghe porzioni dello spazio di ricerca vengono scartate.

## 7.5 IBM Neural Network Utility

Neural Network Utility (NNU) è un sistema sviluppato dall'IBM per lo sviluppo di applicazioni data mining basato sulle reti neurali [3]. Il sistema fornisce la possibilità di usare diversi modelli di reti neurali come le reti Multilayer, le Adaptive Resonance Networks, le Self-Organizing Future Maps e le reti Temporal Difference Learning. Quindi il sistema NNU offre agli utenti la possibilità di definire in maniera semi-automatica diversi algoritmi di data mining in base ai dati di input ed al problema da risolvere. Inoltre fornisce strumenti per la preparazione dei dati e strumenti per l'analisi grafica dei dati di input e per la visualizzazione dei dati di output. È anche possibile visualizzare i dati intermedi, i parametri della rete durante l'uso e le regole che vengono usate dalla rete per il data mining.

NNU permette di avere diversi formati di input: file, fogli elettronici o database, ed assiste l'utente durante la fase di preparazione dei dati per la loro pulizia, ad esempio, tramite il loro filtraggio, la sostituzione di simboli e l'eliminazione di valori numerici non interessanti usando dei valori di soglia. NNU permette anche di definire regole *fuzzy* da combinare con le reti neurali per l'elaborazione della conoscenza. Inoltre NNU assiste l'utente nella fase di training e testing delle reti neurali usate per le applicazioni di data mining.A questo scopo NNU dispone di uno *scripting language* che permette di definire in maniera strutturata (tramite costrutti $if - then - else$, $while$, etc.) le fasi di training delle reti scelte.

A titolo di esempio, vediamo brevemente come tramite NNU può essere sviluppata una applicazione di data mining. L'applicazione qui considerata riguarda la segmentazione di un particolare settore di mercato: i dati sui clienti, i prodotti, e gli acquisti dei clienti vengono aggregati e selezionati per ottenere una "vista" delle caratteristiche dei clienti ed dei loro comportamenti rispetto a quattro categorie di prodotti. Il modello di rete neurale usata è la Self-Organizing Future Map.

La prima operazione da fare è selezionare i dati che devono essere usati per la segmentazione e scegliere una loro rappresentazione. Per effettuare il clustering dei dati con una rete neurale, questi vengono normalizzati e riportati in un intervallo, ad esempio da 0 a 1. A questo punto le rete neurale viene usata per segmentare i dati dei clienti in un insieme di gruppi target in base ad alcuni comportamenti e caratteristiche desiderate. I vari gruppi così ottenuti possono a loro volta essere segmentati per analisi di maggiore dettaglio. Alla fine tutti i clienti verranno classificati ottenendo dei gruppi di clienti per ognuno delle quattro categorie di prodotti target.

Il sistema NNU è stato uno dei primi prodotti realizzati per l'uso delle reti neurali nel settore del data mining. NNU è disponibile su diverse architetture, a partire dai personal computer (PC) fino al sistema parallelo IBM SP2. Inoltre è possibile utilizzare il sistema su una rete di calcolatori secondo la modalità client/server, in cui i PC possono agire da nodi clienti e le macchine più potenti agiscono da server. Il sistema NNU è stato usato per realizzare applicazioni di data mining in vari settori, come ad esempio la definizione di target di mercato, la stima di prezzi, e la previsione di vendita in particolari settori di mercato.

## 7.6 GA-MINER

GA-MINER è un sistema basato sull'uso degli algoritmi genetici [16] per il data mining su basi di dati di notevoli dimensioni.

Nell'utilizzare gli algoritmi genetici il sistema GA-MINER usa l'approccio Pittsburgh secondo cui ogni individuo o entità della popolazione è un insieme di regole che rappresentano una completa soluzione al problema. L'algoritmo genetico usato in GA-MINER usa un modello di popolazione strutturata in cui ogni partner riproduttivo di un genoma è selezionato nell'ambito del suo vicinato. Gli operatori di *crossover* e *mutazione* sono definiti a differenti livelli sulla base della struttura della rappresentazione. Clausole, termini e valori degli attributi rappresentano i singoli geni nei diversi livelli di rappresentazione. Durante la sua esecuzione, GA-MINER colleziona insiemi di pattern. L'aggiornamento dell'insieme dei pattern è effettuato dal sistema usando una semplice euristica basata sulla sostituzione della regola che si adatta peggio o della regola più simile (se la similarità supera un dato valore di soglia) con la regola che si adatta meglio.

Sebbene il sistema possa anche essere utilizzato come strumento di classificazione di dati, esso è stato definito per essere uno strumento generale per la ricerca di pattern usando diversi livelli di supervisione degli utenti. Infatti, un elemento centrale di GA-MINER è il *pattern template*, che definisce la forma generale dei pattern di interesse dell'utente e restringe lo spazio di ricerca ai pattern che sono consistenti con la forma definita. Le componenti di un pattern template possono essere *initialized* o *fixed*. Le parti *fixed* vengono ereditate da ogni pattern nella popolazione e non vengono mai modificati dalle operazioni di crossover e mutazione. Le parti *initialized* del template possono essere modificate durante la ricerca. Nel caso in cui si vuole usare un basso livello di supervisione da parte dell'utente occorre definire un template minimo.

GA-MINER supporta diversi modelli di pattern, come pattern di correlazione, pattern con regole esplicite e *distribution shift patterns*. Tutte queste forme sono basate sulla *subset description*. Le *subset description* sono clausole che vengono usate per selezionare sottoinsiemi in un database e costituiscono gli oggetti principali che sono manipolati dall'algoritmo genetico. I pattern sono costruiti come interpretazioni di più alto livello di un certo numero di *subset*. Ad esempio, un pattern di correlazione potrebbe essere espresso come

**when** $C$ the variables $A$ **and** $B$ are correlated.

Un pattern con regole esplicite può usare due *subset description* $A$ e $B$ che rappresentano rispettivamente la condizione e la predizione in una regola del tipo

**if** $A$ **then** $B$.

Infine, un *distribution shift pattern* può essere definito nella forma seguente:

The distribution of $A$ **when** $B$ **and** $C$ is significantly different from

the distribution of $A$ **when** $C$.

Si noti che, poichè la stessa rappresentazione (*subset description*) è usata per tutti i modelli di pattern, potrà essere utilizzato lo stesso algoritmo genetico per tutti i pattern.

GA-MINER è stato sviluppato in Inghilterra della Quadstone Ltd ed è stato usato, ad esempio, per l'analisi di dati forniti dalla Barclays Bank e dalla GMAP Ltd. GA-MINER è stato implementato usando il linguaggio $RPL2$ (Reproductive Plan Language) su macchine parallele sia di tipo SMP (Symmetric Multiprocessor) sia di tipo MPP (Massively Parallel Processor).Un esempio di elaboratore appartenente alla classe MPP su cui è stato implementato GA-MINER è il CRAY T3D.

Dal punto di vista delle prestazioni, il sistema si è dimostrato scalabile all'aumentare del numero dei processori usati. Dal punto di vista dell'accuratezza dell'analisi, GA-MINER si è mostrato molto utile sia per la classificazione sia per il *pattern discovery*. In particolare GA-MINER è stato usato con risultati positivi per la scoperta di pattern senza la guida dell'utente (*undirect data minig*) e quindi per trovare pattern inattesi senza possibili pregiudizi da parte dell'utente. Si noti comunque che il sistema supporta anche il *direct data minig* in cui l'utente fa delle richieste specifiche usando un pattern template complesso, ed l'*hypotesis testing and refinenement* in cui l'utente fa delle ipotesi inserendo anche delle componenti random nei pattern, quindi chiede al sistema se esse sono vere ed eventualmente di affinarle.

# 8 Data Mining e Calcolo Parallelo

Esistono oramai database composti da diversi milioni di record la cui ricerca richiede algoritmi efficienti e con alte prestazioni. Per analizzare *efficientemente* database composti da gigabyte o terabyte di dati, molte aziende o enti si stanno dotando di sistemi di calcolo paralleli che permettono di ottenere risposte in tempi rapidi. L'utilizzo dei sistemi di calcolo parallelo, oltre a permettere una più veloce esecuzione delle applicazioni rispetto all'esecuzione sequenziale, in molti casi permette di effettuare delle ricerche ed eseguire delle query che non possono essere eseguite usando le architetture di calcolo sequenziali.

Per queste ragioni l'utilizzo di tecniche e architetture a parallelismo massiccio assume una grande importanza per realizzare sistemi di knowledge discovery e data mining in database [35]. In particolare, il parallelismo può portare notevoli benefici nell' implementazione di applicazioni di data mining sia perchè permette di realizzare algoritmi di data mining paralleli, sia perchè permette di implementare in maniera efficiente i DBMS sfruttando il parallelismo presente nell' esecuzione delle query [30].

Tra i passi funzionali di cui è composto il processo di data mining vi sono l'estrazione dei pattern, l'uso dei pattern estratti per costruire modelli, e la selezione o combinazione di più modelli per produrre una decisione. Questi passi operativi possono essere effettuati secondo diversi metodi, i quali per operare devono scegliere uno o più sottoinsiemi dei dati interessati dai quali estrarre le informazioni. I metodi variano a seconda del modo in cui i dati sono selezionati e analizzati: i sottoinsiemi possono essere creati tramite un partizionamento dei dati, attraverso una campionamento *random*, da una copertura dei dati (*cover-based data mining*), o tramite una *range-based query* sui dati. Ognuno di questi metodi permettono di esplicitare un certo grado di parallelismo che in alcuni casi è significativo, mentre in altri lo è meno.

Nel processo di data mining basato sul partizionamento dei dati, l'insieme dei dati viene partizionato in sottoinsiemi distinti e quindi su ogni partizione viene effettutata l'estrazione dei pattern e il calcolo del modello. Quindi tutti i modelli cosi ottenuti vengono combinati allo scopo di selezione un modello. Come si può intuire questa tecnica di data mining può essere facilmente parallelizzata eseguendo in parallelo le operazioni di estrazione e calcolo di un modello su ogni singola partizione. In questo caso il parallelismo esplicitabile è detto *data parallelism* e permette di eseguire in parallelo tanti processi per quante sono le partizione del data set.

Il data mining basato sul metodo del campionamento è quello più comunemente usato. Si effettua una campionamento in un grande insieme di dati e quindi si estraggono dei pattern dai dati selezionati per costruire un modello. Questo metodo ha il vantaggio di ridurre notevolmente i tempi di elaborazione poichè lavora solo sui dati campione. D'altra parte esso presenta lo svantaggio di ottenere, in alcuni casi, dei pattern poco rappresentativi dell'intero set dei dati. Questo metodo, di solito, non viene parallelizzato, anche se in alcuni casi può essere utile effettuare la fase di campionamento ed estrazione dei pattern in parallelo per poi combinare i risultati ottenuti dai varie attività svolte in parallelo allo scopo di calcolare il modello.

Il cover-based data mining è simile al data mining basato sul partizionamento dei dati. Ma in questo caso i sottoinsiemi o partizioni possono avere delle sovrapposizioni, cioè si possono avere dati replicati su più partizioni. Questo approccio presenta delle similitudini con l'approccio *locale*, in cui i pattern estratti usano dati che presentano delle caratteristiche di località. Ad esempio, usano i dati che presentano una distanza minore di $N$ rispetto ad un dato usato come riferimento. Come il metodo basato sul partizionamento, anche questo metodo può essere parallelizzabile con risultati significativi in termini di performance.

Il *range-based query* data mining crea differenti sottoinsiemi dei dati da analizzare sulla base dell'esecuzione di query in cui vengono usati dei particolari attributi rispetto a cui valori viene operata la selezione. Ad esempio, si possono ottenere le partizioni in cui il primo attributo ha valore "$X$" ed il secondo attributo ha valore maggiore di 10. Dopo che le partizioni sono state ottenute esse possono essere analizzate per estrarre i pattern. Anche in questo caso, quando le partizioni sono state definite è possibile eseguire l'estrazione dei pattern e del modello eseguendo queste operazioni in parallelo su ogni partizione ottenendo un notevole risparmio di tempo.

Oltre ai classici approcci, un nuovo approccio che sembra molto promettente è basato sull'uso concorrente di differenti tecniche di analisi dei dati applicate su partizioni di un database o su differenti siti. Dopo aver ottenuto i risultati, questi possono essere integrati per ottenere un modello differente e possibilmente migliore dei modelli ottenuti tramite le differenti analisi effettuate in parallelo. In questo caso il calcolo parallelo non serve solo ad ottenere migliori prestazioni, ma anche a migliorare l'accuratezza dei risultati ottenuti nel processo di data mining.

Attraverso l'uso integrato dei calcolatori paralleli e di Internet è anche possibile realizzare dei sistemi paralleli per il data mining distribuiti su tutto il territorio mondiale. Secondo questo approccio un certo numero di processi che eseguono applicazioni di knowledge discovery sul proprio database locale possono essere eseguiti su macchine parallele o workstation remote per effettuare delle analisi concorrentemente e quindi i risultati ottenuti in ogni singolo sito possono essere raccolti e fusi tramite Internet. In questo modo un insieme di macchine connesse tramite Internet che eseguono concorrentemente una applicazione di data mining possono essere viste come un singolo calcolatore massicciamente parallelo composto da migliaia di nodi di elaorazione.

## 8.1 Sistemi Data Mining Paralleli

Negli ultimi anni sono stati realizzati sistemi, strumenti ed algoritmi di knowledge discovery e data mining utilizzando architetture di calcolo parallelo allo scopo di ottenere sistemi di data mining ad alte prestazioni. Di seguito sono citati solo alcuni esempi significativi di applicazioni di data mining implementate su elaboratori paralleli.

- Il Dipartimento del Tesoro degli USA utilizza FAIS (FinCEN Artificial Intelligence System), una applicazione di data mining, implementata dalla FinCEN Agency su un server parallelo Sun con 6 processori Sparc. FAIS [18] ha lo scopo di controllare eventuali casi di riciclaggio di denaro "sporco" tramite l'analisi in grandi database delle transazioni effettuate da banche e casinò. L'ordine di grandezza delle transazioni analizzate è di 20 milioni con centinaia di attributi per ogni transazione. Per ottenere delle prestazioni accettabili l'applicazione è stata implementata in parallelo tramite un insieme di processi che in maniera asincrona implementano le varie operazioni di knowledge discovery.

- L'AT&T Global Information Systems ha lo scopo di realizzare una serie di applicazioni di data mining detta Solution Packs. Le Solution Packs forniscono una soluzione in vari settori applicativi come i sistemi finanziari, i trasporti, le comunicazioni, la fabbricazione e la vendita al dettaglio di beni di consumo. Una Solution Pack può essere eseguita su macchine parallele di diverso tipo come SMP, cluster di SMP, e sistemi a parallelismo massiccio.

- La HP ha sviluppato il sistema Information Discovery-IDIS per il data mining su sistemi Open Parallel Unisys Server (OPUS) che utilizzano Oracle. IDIS analizza una base di dati, genera una serie di ipotesi e in base a queste tenta di scoprire patterns a regole di conoscenza. IDIS è stato utilizzato in molte area applicative come controllo della qualità, previsioni finanziarie, prevenzione di incendi, diagnostica medica, e applicazioni bancarie.

- Il sistema IRS esegue applicazioni di data mining per migliorare il controllo del pagamento delle tasse in USA e rilevare eventuali frodi fiscali. Il sistema è stato installato in molte sedi nel territorio degli USA e viene usato per analizzare complessivamente circa 100 milioni di transazioni al mese.

- Il Jet Propulsion Laboratory della NASA usa tecniche di data mining per analizzare i dati raccolti dal progetto Palomar Sky Survey [13]. Tramite l'uso di queste tecniche nel sistema SKICAT sono state scoperte 9 nuove quasar. In questo stesso settore applicativo, nell'ambito del progetto SETI (Search for Extraterrestrial Intelligence) si sta cercando usare tecniche di data mining utilizzando una rete di migliaia di calcolatori che lavorano in parallelo per filtrare la grande mole di dati che vengono ricevuti dal grande radio telescopio di Arecibo in Portorico.

- La catena di negozi americana Wal-Mart con 2000 punti vendita usa una macchina a parallelismo massiccio Teradata per analizzare i dati degli acquisti individuali dei clienti che corrispondono ad alcuni terabytes con un tasso di crescita di circa 20 milioni di transizioni al giorno.

- L'AT&T usa applicazioni di data mining per l'analisi i dati della propria clientela utilizzando più di 50 calcolatori Teradata con oltre 3000 dischi.

- La catena di negozi tedesca Karstadt usa un sistema neurale parallelo sviluppato dalla Neurotec per la predizione delle vendite di circa 200.000 prodotti allo scopo ottimizzare gli ordini.

- La società americana Epsilon Data Management usa una applicazione di data mining su di una macchina parallela come la Connection Machine per analizzare i dati dei clienti della American Express allo scopo di ottimizzare le campagne pubblicitarie fatte per posta dell'American Express.

- La società Incyte Pharmaceuticals Inc. ha realizzato un sistema di data mining chimato LifeTools 3D, basato sull'ambiente software MineSet(tm) sviluppato dalla Silicon Graphics. LifeTools 3D permette l'applicazione di tecniche di data mining in campo medico e farmaceutico e può essere utilizzato sulle macchine parallele della Silicon Graphics.

- In Inghilterra, la banca Abbey National usa un sistema parallelo Teradata per l'analisi dei dati dei clienti e per ottimizzare le campagne pubblicitarie postali.

- DB-INSPECTOR è un sistema per l'analisi di dati bancari allo scopo di scoprire attività finanziarie fraudolente. Questo sistema è stato sviluppato nell'ambito di un progetto europeo ESPRIT svolto nel settore HPCN (High Performance Computing and Networking). DB-INSPECTOR utilizza tecniche di data mining basate su reti neurali per l'analisi di basi di dati di tipo finanziario. Esso è stato implementato su una macchina AT&T GIS 3600 utilizzando Oracle v7 RDBMS con Parallel Server per l'esecuzione parallela delle query sul database.

Occorre infine sottolineare che oltre ai sistemi qui citati, le varie case costruttrici di macchine parallele e di software parallelo hanno già sviluppato o stanno sviluppando sistemi ed applicazioni parallele di data mining e knowledge discovery. Alcuni esempi sono rappresentati dai prodotti sviluppati da IBM (Intelligent Miner), Thinking Machines, e Silicon Graphics (MineSet).

# 9 Conclusioni

Le techniche ed i sistemi di data mining avranno nei prossimi anni un forte impatto nell'analisi e la gestione di grandi moli di dati. Questo lavoro fornisce una introduzione alle tecniche di data mining e knowledge discovery in basi di dati ed analizza alcuni sistemi di $KDD$ e data mining che sono stati sviluppati in questi ultimi anni. L'obiettivo era quello di fornire un'introduzione alle tecniche di data mining e di mostrare come le applicazioni di data mining e knowledge discovery nelle basi di dati possano essere di notevole beneficio in molti settori applicativi.

Naturalmente esistono numerosi aspetti del data mining non ancora del tutto chiariti e numerosi problemi aperti sia dal punto di vista della teoria generale del data mining sia per ciò che riguarda l'efficienza computazionale degli algoritmi e delle architetture usate [25]. Nonostante ciò il data mining rappresenta un settore in grande sviluppo per l'analisi e l'estrazioni di dati in molte aree applicative.

Infatti, gli analisti del META Group, una società statunitense specializzata nell'analisi del mercato delle tecnologie dell'informazione, prevedono che, ad esempio, nell'ambito del mercato dei decision support systems il segmento data mining avrà un tasso di crescita annuale di oltre il 250% passando dai 120 milioni di dollari nel 1996 a circa 800 milioni di dollari nel 1998 e supererà i 4 miliardi di dollari entro il 2000. In accordo con quanto riportato dal META Group, un recente studio del Gartner Group riporta il data mining in cima alla lista delle aree tecnologiche di interesse aziendale ed afferma, inoltre, che nei prossimi 3-5 anni esso assumerà sempre maggiore importanza nel mondo commerciale e finanziario.

# References


[1] Babcock, C., Parallel Processing Mines Retail Data. In *Computer World*, 6, 1994.

[2] Bergadano, F., Gunetti, D. *Inductive logic programming: fomr machine learning to software engineering.* MIT Press, Cambridge, 1995.

[3] Bigus, J.P., *Data Mining with Neural Networks.* McGraw-Hill, New York, 1996.

[4] Charniak, E., McDermott, D., *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA, 1985.

[5] Chen, M.-S., Han, J., Yu, P.S., Data Mining: An Overview from a Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), pp. 866-883, 1996.

[6] Compton, P., Jansen, R., Knowledge in Context: A Strategy for Expert System Maintenance. In *Proc. 2nd Australian Join Art. Int. Conf., LNAI 406*, pp. 292-306, Adelaide, 1988.

[7] De Raedt, L., *Advances in Inductive Logic Programming.* IOS Press, 1996.

[8] Dimopoulos, Y., Kakas A., Abduction and Learning. In *Advances in ILP*, De Raedt (ed.), IOS Press, 1996.

[9] Duda, R., Hart, P., *Pattern Classification and Scene Analysis.* New York, Wiley, 1973.

[10] Fahlman, S.E., Hinton G.E., Connectionist Architectures for Artificial Intelligence. *Computer*, 20, pp. 100-109, IEEE CS Press, Gen. 1987.

[11] Fasman, K.H., Cuticchia, A.J., Kingsbury, D.T., The GDB (TM) Human Genoma Database. In *Nucl. Acid. R.*, N. 22(17), pp. 3462-3469, 1994.

[12] Fayyad, U.M., Djorgovski, S.G., Weir, N., Automating the Analysis and Cataloging of Sky Surveys. In *Advances in Knowledge Discovery and Data Mining*, pp. 471-494, AAAI MIT Press, 1996.

[13] Fayyad, U.M., Haussler, D., Stolorz, P., KDD for Science Data Analysis: Issues and Examples. In *Proc. KDD-96 -2nd Int. Conf. on Knowledge Discovery & Data Mining*, pp. 50-56, AAAI Press, 1996.

[14] Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P., From Data Mining to Knowledge Discovery: an Overview. In *Advances in Knowledge Discovery and Data Mining*, pp. 1-34, AAAI MIT Press, 1996.

[15] Fayyad, U.M., Uthurusamy, R., Eds., Data Mining and Knowledge Discovery in Databases. *Communications of the ACM*, 39(11), 1997.

[16] Flockart, I.W., Radcliffe N.J., A Genetic Algorithm-Based Approach to Data Mining. *Proc. KDD-96 - 2nd Int. Conf. On Knowledge Discovery and Data Mining*, AAAI Press, pp. 299-302, 1996.

[17] Fogelman Soulie, F., Neural Networks and Computing. In *Future Generation Computer Systems*, 7, pp. 69-77, 1991.

[18] Goldberg, H.G., Senator, T.E., Restructuring for Knowledge Discovery by Consolidation and Link Formation. In *Proc. KDD-95 - 1st Int. Conf. on Knowledge Discovery & Data Mining*, pp. 136-141, Montreal, 1995.

[19] Holland, J.H., Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan, 1975.

[20] Holland, J.H., Escaping Brittleness : The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. *Machine Learning : an Artificial Intelligence Approach*, 2, 1986.

[21] Holsheimer, M., Siebes, A.P. Data Mining: the Search for Knowledge in Databases. Tech. Rep. CS-R9406, Dept. of Computer Science, CWI, Amsterdam, NL, 1994.

[22] James, M., *Classification Algorithms.* New York, Wiley, 1985.

[23] Knight, K., Connectionist Ideas and Algorithms. *Communications of the ACM*, vol. 33, pp. 59-74, Nov. 1990.

[24] Lavrac, N., Dzeroski, S., *Inductive Logic Programming: Tecniques and Applications.* Hellis Horwood, 1994.

[25] Mannila, H., Methods and Problems in Data Mining. *Proc. Int. conf. on Database Theory*, Delphi, Greece, Springer-Verlag, 1997.

[26] McClelland, W.S., Rumelhart, D.E., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* The MIT Press, 1986. (volumes 1 and 2)

[27] Michalski,R.S.,Mozetic, I., Hong, J., Lavrac, N. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. of the 5th National Conference on Artificial Intelligence*, pp. 1041-1045, Philadelphia, 1986.

[28] Muggleton, S. (eds), *Inductive Logic Programming.* Academic Press, 1992.

[29] Nilsson, N.J., *Principles of Artificial Intelligence.* Symbolic Computation, Springer-Verlag, 1982.

[30] Ozsu, M.T., Valduriez, P., Distributed and Parallel Database Systems. *ACM Computing Surveys*, vol. 28, no. 1, pp. 125-128, 1996.

[31] Quinlan, J.R., Induction of Decision Trees. In *Machine Learning*, 1, pp. 81-106, 1986.

[32] Quinlan, J.R., Generating Production Rules from Decision Trees. In *Proc. Int. Conf. on Artificial Intelligence*, pp. 304-307, Milano, 1987.

[33] Quinlan, J.R., Learning Logical Definitions from Relations. In *Machine Learning*, 5, pp. 239-266, 1990.

[34] Quinlan, J.R., Knowledge Acquisition fro Structured Data. In *IEEE Expert*, 6(6), pp. 32-37,1991.

[35] Reese Hedberg, S., Parallelism Speeds Data Mining. *IEEE Parallel & Distributed Technology*, vol. 2, no. 4, pp. 3-6, 1995.

[36] Rivest, R.L., Learning Decision Lists. In *Machine Learning*, 2, pp. 229-246, 1987.

[37] Schlimmer, J., Armstrong, G., Little, J.D.C., CoverStory - automated news finding in marketing. In *DSS Transactions*, Institute on Management Sciences, Providence, RI, 1996.

[38] Shapiro, R.L., *Algorithmic Program Debugging.* Cambridge, MA, The MIT Press, 1983.

[39] Simpson, P.K., *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations.* Pergamon Press, 1990.

[40] Smith, S.F., *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[41] Smith, S.F., Adaptive Learning Systems. *Expert Systems, Principles and Case Studies*, Chapman and Hall, 1984.

[42] Talia, D., Reti Neurali: Modelli ed Algoritmi. *Rivista di Informatica*, Masson, vol. XXII, no. 2, pp. 127-144, 1993.

[43] Ullman, J.D., *Principles of Database and Knowledge-base Systems.* vol. 2, Computer Science Press, 1989.

[44] Wasserman, P.D., *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, 1989.

[45] Way, J., Smith, E.A., The Evolution of Synthetic Aperture Radar Systems and their Progression to the EOS SAR. In *IEEE Trans. on Geoscience and Remote Sensing*, 29(6), pp. 962-965, 1991.

[46] Widrow, B., Lehr, M.A., 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE*, 78(9), pp. 1415-1442, IEEE Press, 1990.

[47] Willshaw, D., Neural Systems and Models. *Proceedings European Seminar on Neural Computing*, IBC, London, Feb. 1988